Technical Report 1089

A Theory of Quantitative Inference Applied to a Mechanical Design Compiler

Allen C. Ward

MIT Artificial Intelligence Laboratory

This blank page was inserted to preserve pagination.

A Theory of Quantitative Inference for Artifact Sets, Applied to a Mechanical Design Compiler

by

Allen Corlies Ward

S.B. University of Oregon (1973)
S.B. University of Hawaii (1981)
S.M. Massachusetts Institute of Technology (1984)

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor Of Science

at the
Massachusetts Institute Of Technology
January 1989

©Massachusetts Institute of Technology 1988. All rights reserved.

Signature of Author Allen C. Ward

Allen Corlies Ward
Department of Mechanical Engineering
January 31, 1989

Certified by

Professor Warren P. Seering
Thesis Supervisor

Accepted by

Ain A. Sonin
Chairman, Departmental Graduate Committee

This empty page was substituted for a blank page in the original document.

A Theory of Quantitative Inference for Artifact Sets Applied to a Mechanical Design Compiler

by

Allen Corlies Ward

Submitted to the Department of Mechanical Engineering on January 31, 1989 in partial fulfillment of the requirements for the degree of Doctor of Science in Mechanical Engineering.

Abstract

This thesis presents the ideas underlying a computer program that takes as input a schematic of a mechanical or hydraulic power transmission system, plus specifications and a utility function, and returns catalog numbers from predefined catalogs for the optimal selection of components implementing the design. Unlike programs for designing single components or systems, the program provides the designer with a high level "language" in which to compose new designs. It then performs some of the detailed design process.

The process of "compilation", or transformation from a high to a low level description, is based on a formalization of quantitative inferences about hierarchically organized sets of artifacts and operating conditions. This allows design compilation without the exhaustive enumeration of alternatives.

The program has been tested on a wide variety of power transmission and temperature sensing problems. Key elements of the theory have been formally proven. It appears that the theory has applications outside design.

Thesis Supervisor: Dr. Warren Seering
Title: Associate Professor of Mechanical Engineering

This empty page was substituted for a blank page in the original document.

Acknowledgments

Many people contributed to the work described in this thesis. One stands out: my thesis advisor, Warren Seering, supported my investigation of an open-ended and unusual research topic, tolerated my frequent pursuit of competing interests, and taught me to communicate my ideas; without his extraordinary generosity and enthusiasm the entire project would have been impossible. Tomás Lozano-Pérez patiently explained fundamental computer science. Woodie Flowers helped keep me focused on reality. David McCallister provided a key insight. My office mates, Ken Pasch, Neil Singer, and Erik Vaaler, and fellow students, Mike Caine, Andy Christian, Steve Eppinger, Steve Gordon, Bob Hall, Peter Meckl, Lukas Ruecker, Kamala Sundaram, Bruce Thompson, and Karl Ulrich, provided years of companionship, argument, and encouragement.

My wife Yasuko supported me financially and emotionally; with my son Henry, she made the work worthwhile. My parents, Carol and Marshall, died during the course of this work; they always encouraged me to ask questions, and were proud when I occasionally found answers.

This research was performed at the MIT Artificial Intelligence Laboratory, whose faculty, staff, and students have created a remarkably supportive research environment.

Funding for the work was provided in part by the Industrial Technology Institute of Ann Arbor, Michigan, and in part by the Office of Naval Research under University Research Initiative contract N00014-86-K-0685.

Contents

1 Introduction **8**
- 1.1 What's this about? 8
- 1.2 Why read this? 10
  - 1.2.1 The short-term, pragmatic argument 10
  - 1.2.2 The basic research argument 12
- 1.3 How to read this thesis 13

2 The Program and its Performance **16**
- 2.1 Introduction 16
- 2.2 Overview of the Compiler 16
- 2.3 Some Examples 21
- 2.4 Assessing Program Reliability 27
- 2.5 Time Complexity 29
  - 2.5.1 Theoretical Results 29
  - 2.5.2 Empirical Results 30
  - 2.5.3 Explaining the difference 31
- 2.6 Conclusions 34

3 The labeled interval calculus **36**
- 3.1 Introduction 36
  - 3.1.1 Preview 38
- 3.2 A design compiler 39
- 3.3 Some Operations on Intervals 41
- 3.4 The labeled interval specification language 45
  - 3.4.1 Limits and operating regions 46
  - 3.4.2 Required, Assured, and No-stronger labels 47
- 3.5 Operations on Labeled Intervals 48

3.5.1 Elimination 48
3.5.2 Abstraction 49
3.5.3 Propagating Labeled Intervals Using Equations 50
3.6 Conclusion 53

4 Extending Constraint Propagation 56
4.1 Introduction 56
4.1.1 An Example 57
4.1.2 Assignment Intervals, and Equations 59
4.2 Interval Operations and Inferences 62
4.2.1 Conventional Constraint Propagation 62
4.2.2 The DOMAIN Operation 66
4.2.3 The Sufficient-Points Operation 70
4.3 Some Application Problems 72
4.3.1 Relaxing the Monotonicity Assumption 73
4.3.2 The Termination Issue 73
4.3.3 Results 77

5 Inferences on Sets of Artifacts 78
5.1 Introduction 78
5.2 Labeled Interval Propagation 81
5.2.1 Extending Single-Artifact Rules to Sets 82
5.2.2 The "No-stronger" Statement 86
5.2.3 The rules 88
5.3 Abstraction and Search 96
5.3.1 Abstraction 96
5.3.2 Elimination and Search 98
5.4 Conclusion 100

6 The Basic Ideas 101
6.1 Introduction 101
6.2 The Ideas 103
6.3 Conclusion 114

7 The context of the work 115
7.1 The Past: A Review of the literature 115
7.2 Future Work 118

7.2.1 The Near Term: Checks and Improvements 118
7.2.2 The Medium Term: Extensions to the System 121
7.2.3 The Long Term: Outside Machine Design 128
7.2.4 Planning 130

A Definitions **139**

List of Figures

1.1 Sample schematic elements 9
1.2 The Organization of the Work 13

2.1 The Compiler Block Diagram 17
2.2 A hierarchy of motors 19
2.3 A Hydraulic Power Train 20
2.4 Some test parts 22
2.5 Some test designs 24
2.6 Specification generation vs alternatives for a variety of designs 32
2.7 Specification generation vs equations for a variety of designs . 33

3.1 The Toscanini's Problem 37
3.2 An illustration of the RANGE operation 43
3.3 An illustration of the DOMAIN operation 44
3.4 An illustration of the SUFFICIENT-POINTS operation 45
3.5 A very simple power train 46

4.1 Inferences on a Mechanical Transmission 58
4.2 A non-terminating propagation net 74
4.3 The "strongly consistent" mechanical transmission equations . 75

5.1 A pair of simple mechanical schematics 79
5.2 "A" ball bearing 80
5.3 Selecting a tolerance 91
5.4 Abstracting labeled intervals 97

This empty page was substituted for a blank page in the original document.

Chapter 1

Introduction

This chapter answers three questions: First, what is this thesis about? Second, why might it be worth reading? Third, how can it be read most easily and effectively?

1.1 What's this about?

This thesis describes the theory underlying, and the performance of, a computer program which selects standard components from catalogs in order to implement a wide variety of mechanical designs. The program's user forms a schematic diagram of the desired design by combining such elements as those in Figure 1.1. He also enters specifications in a special "labeled interval" language, and a cost function to be minimized. The program returns the catalog numbers for an optimal selection of components.

We can view the schematics and the specifications as a description in a

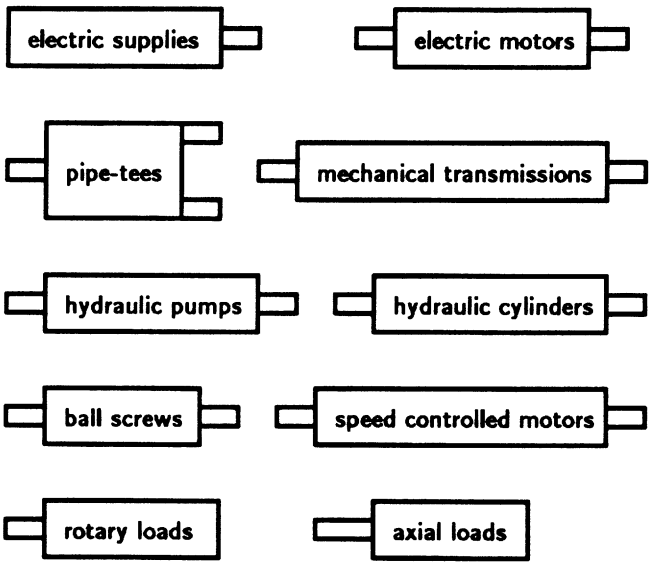


Figure 1.1: Sample schematic elements

high level language, and the catalog numbers as a description in a low-level language. Then, by analogy with computer language compilers, we can call this program a "mechanical design compiler". Like computer language compilers, such programs should improve designer productivity, prevent errors, and allow the exploration of more alternatives in greater depth.

A design compiler is not an "expert system," intended to replace or advise a human designer working on, say, air cylinders. Rather, it is intended to quickly perform part of the designer's task for a broad spectrum of designs, allowing the designer to avoid thinking about some of the details of the design. This objective establishes some fundamental guidelines.

First, the most important details to automate are quantitative; these

are the ones people handle worst, and computers best. Second, compilers should be right all the time; "heuristic" programs can quickly generate more reasonable-sounding but possibly incorrect suggestions than any human can absorb and correct. Third, as I will argue, designers and design compilers must draw inferences about sets of artifacts (physical objects) under sets of operating conditions; they cannot simply simulate or analyze single, completely specified designs. For these reasons, the bulk of this thesis is concerned with the rigorous development of a theory of quantitative inference about sets of artifacts under sets of operating conditions.

1.2 Why read this?

This theory of inference may be a little daunting; its details involve some new notation, a little predicate calculus and set theory, and some reasonably straightforward proofs. It falls squarely on the boundary between mechanical engineering and artificial intelligence research; it therefore requires mechanical engineers to think about informal "designer's intuitions" in a decidedly formal way, and computer scientists to learn something about design. It seems appropriate to indicate why understanding this theory might be worth the trouble.

1.2.1 The short-term, pragmatic argument

This argument comes in two parts. First, mechanical design compilers are potentially powerful, broadly useful engineering tools. My program has been

tested on a wide variety of power transmission system design problems, and a few temperature sensing problems. Even in its current research prototype form it allows me to do the most tedious aspects of such designs an order of magnitude faster than I could unaided.

Second, this compiler appears to be the only one. I have found no other programs identified as "mechanical design compilers" by their creators, but there are some which come close; the ways in which they fall short are informative. The programs of [1] and [2] offer the designer a schematic language, but perform analysis only. They do not select components. The programs of [3] and [4] select components, but do not provide the designer with a schematic language enabling him to quickly formulate new designs. An earlier program of mine [5] provided a schematic language and selected components, but was ad hoc and limited. Finally, Shin-Orr[6], in effect did write a mechanical design compiler, but one limited to the layout of spindle-drive gears for multi-spindle lathes.

I believe that these programs cannot compile a varieties of designs because their representation of specifications is impoverished; they use "constraints" which restrict variables to a certain set of values. In fact, such restrictions are only one of many important relationships connecting variables, artifact sets, value sets, and operating condition sets. The core of this thesis is an analysis of these relationships and their use in design.

1.2.2 The basic research argument

We have a wide variety of mathematical tools for reasoning about a single existing artifact. This thesis provides formal[1] tools for drawing inferences about many possible artifacts and operating conditions simultaneously. Designers have always done this kind of reasoning, just as people can do arithmetic without knowing the associative, commutative, and distributive. But formally expressing such laws provides us with great power to automate, to prove, to generalize, and to explore further.

Arithmetic laws say something fundamental about the way we should think about numbers. Boolean algebra says something fundamental about the way we should think about "True" and "False"[2]; I hope the "labeled interval calculus" says something fundamental about the way we should think about sets of artifacts and operating conditions. Such reasoning is likely to be useful well beyond component selection, and even outside design; Chapter 7 offers some speculations about such uses.

Such formal representations of "ways of thinking" are core pursuits of artificial intelligence research. Indeed, my formalisms are probably too complex for routine human use[3]. They are made useful by symbolic computation; they are necessary because computers are relentlessly formal.

[1] By formal, I mean that they allow correct inferences based only on the *form* of the input expressions, without regard to their meaning; formal operations can be programmed.

[2] Boole called his work "the laws of thought", which seems to go a little too far; I briefly considered titling this thesis "The Laws of Thought, Part N".

[3] This may explain why they were not developed in the nineteenth century.

1.3 How to read this thesis

My work involves a number of different levels (Figure 1.3); each level is addressed by at least one chapter of its own. These chapters can be read in virtually any order; this section is designed to allow readers to pick an appealing sequence. Further, this thesis is a snap-shot of work still very much alive and developing. Some chapters look more durable than others, and this section provides the reader with some warnings in this regard.

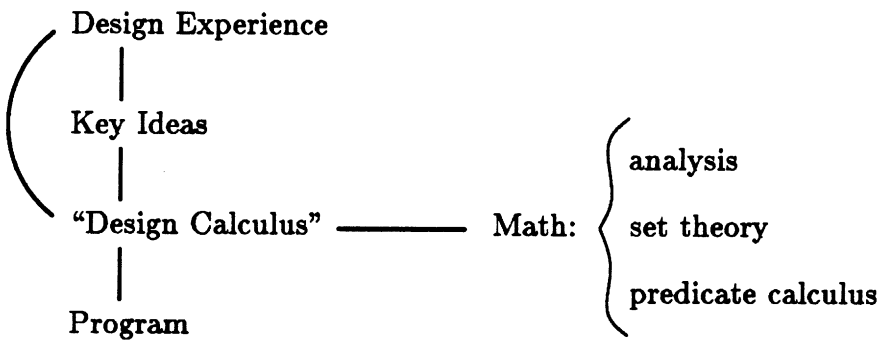


Figure 1.2: The Organization of the Work

Chapter 2 provides an overview of the central algorithm, some sense of what the program is like to use, and a discussion of its performance. It is first in the sequence because I think before investing much time understanding the work, most readers should assure themselves that it is actually good for something. Readers content with a top-level view might read only chapters 2 and 6.1. The algorithm seems sound, but is subject to generalization; the performance results should only be improved by the simpler and more

powerful next generation of design compilers.

Chapter 3 uses design examples to introduce the "labeled interval calculus", my notation and operations for quantitative reasoning about sets of artifacts under sets of operating conditions. It should appeal to mechanical designers who are interested in getting a feel for the system; mathematically inclined readers may prefer to start with the more rigorous approach of chapters 4 and 5. Recent work, too incomplete to present here, suggests that some simplification of the system of labels is possible; these simplifications will change the details of the inference rules presented in this chapter, but not the over-all flavor.

Chapter 4 rigorously develops part of the labeled interval calculus, proving inference rules for single artifacts under varying operating conditions. This chapter seems likely to endure intact any foreseeable improvements.

Chapter 5 extends the formal development to inferences about sets of artifacts. This chapter will be most strongly affected by the developments expected in the near future.

Chapter 6.1 steps back from the technical details to discuss the underlying ideas at a conceptual level. It uses these ideas to compare my work with some closely related work.

Chapter 7 places the work in the context of design research, broadly defined, and speculates on further developments.

Tables A and A at the end of the thesis, lists the key definitions. The reader is encourage to copy them to keep in view while reading the more technical sections.

Two further caveats are in order. The notation used in Chapter 3 is a

little different from that of Chapter 4 and 5, because I wanted to emphasize different things. The inference rules described are also a little different, because I've proved some rules the compiler hasn't tested, and tested some rules I haven't proved.

I have not included a "claim of contributions" chapter, though individual chapters, especially Chapter 6, point out the differences between my work and that of others. Instead, I have tried to clearly label the few places I review previously developed theory; in general, what I have to say here is my own.

Chapter 2

The Program and its Performance

2.1 Introduction

This chapter provides a brief overview of the compiler, then examines its performance in three different respects: 1) the range of design problems it has been tested on; 2) its reliability; and 3) its efficiency or time complexity.

2.2 Overview of the Compiler

Figure 2.1 illustrates my approach. My data base is built up (block 1) from "basic sets" of artifacts. Each basic set is represented by a single catalog number. The set consists of the individual artifacts one might receive by ordering that catalog number. For example, on ordering Dayton motor

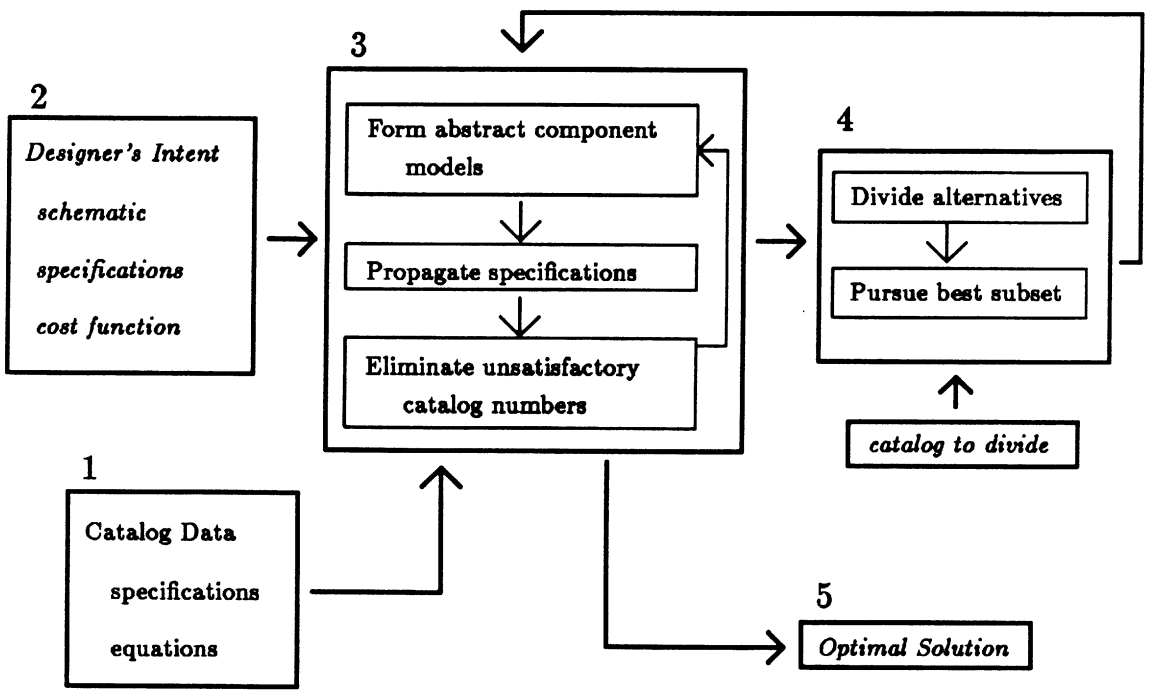


Figure 2.1: The Compiler Block Diagram

number 2N103, we will receive any one of an effectively infinite variety of motors, each slightly different because of manufacturing tolerances, each with its own serial number; these motors make up the basic set denoted by catalog number 2N103.

The basic sets are modeled by an engineer, using equations and specifications in a special "labeled interval" specification language. For example, the speed regulating characteristics of Dayton motors 2N103 might be represented by $\langle \mathbf{A} \overset{only}{[\ ]} \mathit{RPM}\ 1740\ 1800 \rangle$. This specification tells us that we are *assured* (**A**) that for any motor we might get by ordering number 2N103, the the RPM will take on *only* ($\overset{only}{[\ ]}$) values between 1740 and 1800, under normal loading.

The engineer groups the catalog numbers into a hierarchical structure, and the compiler **abstracts** (block 3) the information about the basic motor sets to form descriptions for higher levels in the hierarchy. For example, the next level up might be all the 1800 rpm three-phase motors represented; these have varying degrees of speed regulation, so the set as a whole might only guarantee speed regulation between, say, 1700 and 1800 rpm: $\langle \mathbf{A} \overset{only}{[\ ]} \mathit{RPM}\ 1700\ 1800 \rangle$. Finally, a schematic symbol (Figure 1.1) represents the whole hierarchy of catalog numbers, and therefore the union of their basic sets. The motor symbol might initially represent all of the electric motors listed in the Dayton catalog[1].

The compiler's user, a mechanical designer, **composes** new designs by pointing at schematic symbols (block 2). The system automatically makes

[1]The currently implemented catalogs only include a small subset of the Dayton catalog.

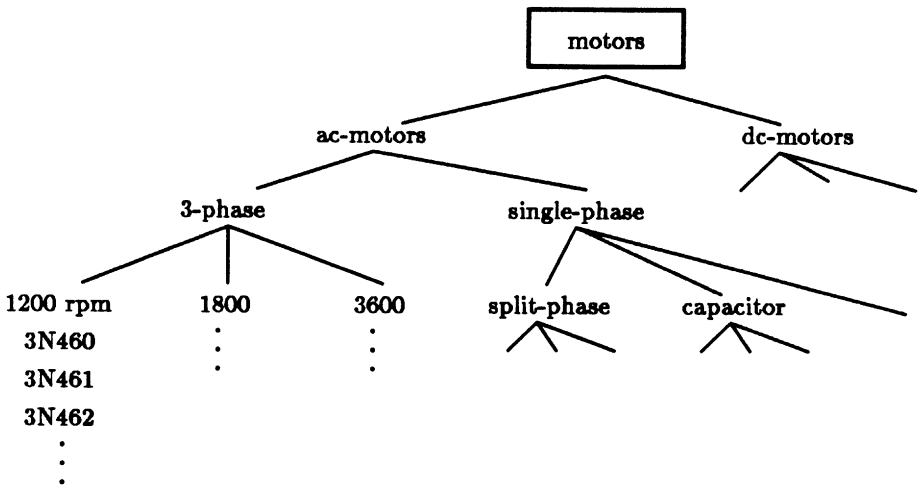


Figure 2.2: A hierarchy of motors

appropriate connections, asking for help if needed to resolve ambiguities; for example, in adding the first cylinder to the schematic of Figure 2.3, the compiler would have to ask which valve to attach it to. Having defined such a design schematic, the user may assign it a symbol of its own, for recall or use in more complex designs.

The compiler automatically **eliminates** catalog numbers which are incompatible with *any* implementation of the connected components (block 3). For example, on connecting the motor schematic to one representing a 220-volt power supply, the system automatically eliminates any 110 volt motors.

After building the schematic, the user provides specifications. These specifications describe sets of operating conditions; $\langle \mathbf{R} \overset{every}{\leftrightarrow} speed\ 0\ .2 \rangle$ applied

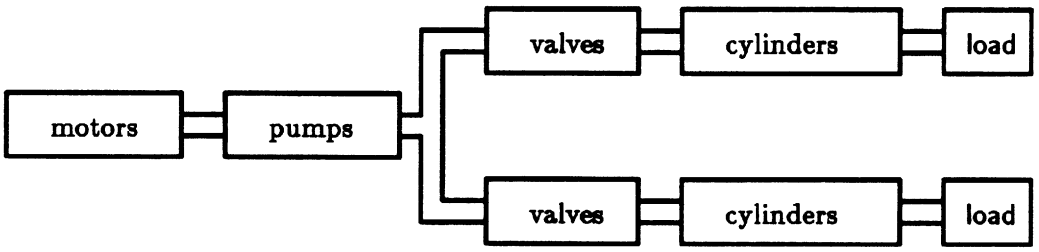


Figure 2.3: A Hydraulic Power Train

to a cylinder means that the speed of the cylinder shaft is "Required" (**R**) to take on every value ($\overset{every}{\leftrightarrow}$) in the interval from 0 to .2 feet per second.

In this example, the maximum output pressure available from any of the pumps, together with the highest of the range of forces required, sets a minimum diameter requirement on the cylinders. These, together with the speeds required, establish flow requirements. This use of equations and specifications to form new specifications is **propagation** (block 3); it can be regarded as a generalization of "the constraint propagation of intervals" [7]. More specifically, the constraint propagation of intervals corresponds to one of 21 propagation operations employed in the compiler.

The propagated specifications for flow, horsepower, torque, and so on cause further eliminations, leaving subsets of the original catalog numbers. Descriptions for these subsets are then abstracted to produce new specifications, which trigger further propagation and elimination.

When the cycle of abstraction, propagation and elimination ceases, a variety of alternative combinations of catalog numbers often remains. The user then provides a cost function, for example the weighted sum of the price

and weight of the components. He also directs the compiler to split one of the catalogs in half, for example to look at 3600 and 1800 rpm motors separately (block 4)[2]. The compiler then generates two daughter designs, one for each motor set; the abstraction operators formulate new specifications describing the new, smaller motor sets. These specifications trigger another cycle of eliminations.

Repeating this splitting process generates a binary best-first search tree. The compiler always splits the leaf of the tree offering the lowest possible cost. The search continues until a single catalog number remained for each component.

The output of this compiler thus consists primarily of catalog numbers. Given these numbers and the schematic, most mechanics could probably buy the components and construct the system without further input from an engineer. A future, more complete compiler would provide a drawing of the base-plate. A yet more complete compiler would instruct an automated manufacturing system to build the design.

2.3 Some Examples

In this section I discuss in general terms my experience with the compiler. Figure 2.4 shows some component types, with the primary variables used to model them.

The components models now used include specifications for most of the

[2]Having the user guide the search in this way improves efficiency; catalogs could be selected for splitting randomly, or by a heuristic.

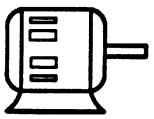
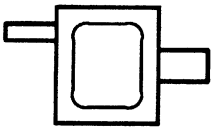
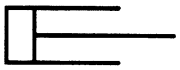
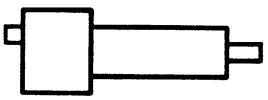
parts	variables represented	
 electric motors	power type voltage current power	rpm efficiency torque
 mechanical transmissions	torque-out torque-in rpm-out rpm-in	power-out power-in efficiency
ball screws	rpm lead speed torque	force critical rpm buckling load length
 hydraulic cylinders	diameter pressure force	flow speed
hydraulic pumps	rpm flow power pressure	displacement total efficiency volumetric efficiency
hydraulic valves	flow-in flow-out pressure-out power-out	valve-coefficient flow-return pressure-in power-in
 speed controlled motors	power type voltage current power	rpm efficiency torque

Figure 2.4: Some test parts

non-geometric criteria that the vendors discuss in the "engineering" sections of their catalogs. Formulating a representation for a component type requires the engineer to extract a precise model, in my formal specification language, from the usually vague and often inconsistent catalog data. He must compromise between simplicity and completeness. For example, I have chosen to represent the efficiency of my mechanical transmissions as a range of possible values, from 90 to 98 percent. I could instead have entered an equation relating efficiency to speed; manufacturing variations would then be represented by interval specifications on the coefficients of that equation.

Once the form of the precise model has been determined, a simple program can be instructed to translate the manufacturer's catalog into the labeled interval specification language. Entering further catalog numbers for components of this type is then a typing exercise. It generally takes about one day to decide the form of the specifications and equations for a new kind of component, generate the transformation procedure that converts the catalog to the desired form, and test the results.

The system has been tested on a few temperature measurement system design problems and more than a dozen different arrangements of power transmission components. Figure 2.5 shows some of these, with machines in which the power trains might be used.

Let us now consider in more detail the two-cylinder hydraulic system example of Figure 2.3. The catalogs for the components shown include the following numbers of alternatives: 7 types of electrical supply (omitted from the schematic), 36 motors, 13 pumps, 3 valves, and 12 cylinders. There are thus 4,245,696 possible combinations; of these, because my catalogs are still

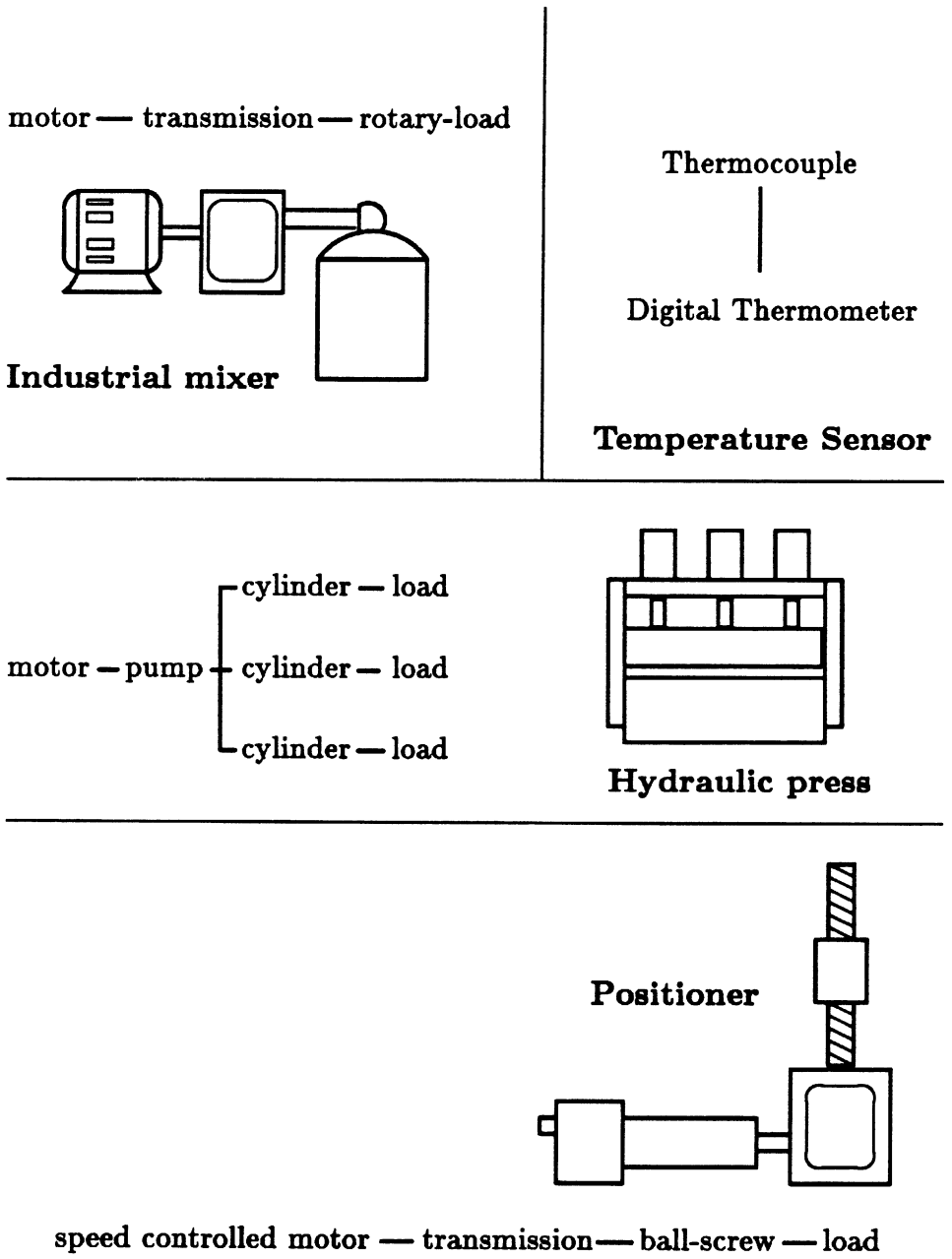


Figure 2.5: Some test designs

sparse, only 505,440 remain after the eliminations caused by connecting the components.

After composing the schematic, the user then enters load specifications, for example:

Load-1: $\langle \mathbf{R} \overset{every}{\leftrightarrow} speed\ 0\ .2\rangle$, $\langle \mathbf{R} \overset{every}{\leftrightarrow} force\ 0\ 1000\rangle$
Load-2: $\langle \mathbf{R} \overset{every}{\leftrightarrow} speed\ 0\ .15\rangle$, $\langle \mathbf{R} \overset{every}{\leftrightarrow} force\ 0\ 3000\rangle$.

For the first load, this means that the system must provide every speed from 0 to .2 feet per second, with forces from 0 to 1000 pounds.

The compiler uses these specifications, and those built into the catalogs, to eliminate unsatisfactory alternatives and to generate further specifications. For example, the linear horsepower equation is built into the "load" component, $hp - \frac{(force)(speed)}{550} = 0$. The compiler incorporates an inference rule which can be written

$$\langle \mathbf{R} \overset{every}{\leftrightarrow} x\ x_l\ x_h\rangle \& \langle \mathbf{R} \overset{every}{\leftrightarrow} y\ y_l\ y_h\rangle \& G(x,y,z) = 0$$
$$\longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow} z\ \textsc{Range}(G, \langle x\ x_l\ x_h\rangle, \langle y\ y_l\ y_h\rangle)\rangle.$$

The left hand side of the rule matches the input data and the equation:

$$\begin{aligned} \langle \mathbf{R} \overset{every}{\leftrightarrow} speed\ 0\ .2\rangle &\sim \langle \mathbf{R} \overset{every}{\leftrightarrow} x\ x_l\ x_h\rangle \\ \langle \mathbf{R} \overset{every}{\leftrightarrow} force\ 0\ 1000\rangle &\sim \langle \mathbf{R} \overset{every}{\leftrightarrow} y\ y_l\ y_h\rangle \\ hp - \tfrac{(force)(speed)}{550} = 0 &\sim g(x,y,z) = 0. \end{aligned}$$

The RANGE function on the right side of the rule is one of three operations on equations and intervals discussed in chapters 3 and 4. In effect, it solves the equation for the hp, forming $hp = \frac{(force)(speed)}{550}$. It then determines the range of the horsepower subject to the constraints that force and speed

are restricted to the intervals [0 1000] and [0 .2]. The numerical results of RANGE are thus identical to those produced by the "constraint propagation of intervals". (The other operations discussed in 3 can be thought of as inverses to RANGE.) But the new specification which would be formulated by the right hand side of the rule , $\langle \mathbf{R} \overset{every}{\leftrightarrows} hp\ 0\ .36 \rangle$, is not a "constraint" in the usual sense of a limit on the values. Rather, it says that the cylinder must have available to it power flows from 0 to .36 horsepower; higher powers are acceptable as well.

These specifications eliminate many potential implementations; for example, motors unable to supply the required horsepower, adjusted by the efficiency of the pumps. The designer then splits the catalog for one of the components, for example one set of cylinders, generating daughter designs. One daughter design has only large cylinders, the other only small; this starts a new cycle of abstraction and elimination.

On the particular data given, the compiler searched 71 daughter designs, generating 15,663 new specifications in the process. The cost function used was price plus one half weight. The design run took about 20 minutes, a normal time for the program to complete a hydraulic problem of this size. Optimization of my code will speed this considerably. The output for this problem included:

```
The optimum solution, with cost 441.97, is:
For POWER-SUPPLY, US-3PH-220 with cost 0
For MOTOR, 3N593 with cost 192.72
For GEAR-PUMP, TYPE-103 with cost 133.0
```

```
For VALVE, TYPE-1 with cost 50.0
For CYLINDER, 1.25 with cost 6.25
For VALVE-2, TYPE-1 with cost 50.0
For CYLINDER-2, diameter 2.0 with cost 10.0
```

2.4 Assessing Program Reliability

I have used basic set theory, predicate calculus, and analysis to develop formal correctness proofs for many of the individual compiler operations; see chapters 4 and 5. Such proofs add greatly to the reliability of the program, and to our understanding, but they are no better than the assumptions on which they rest; the program must still be tested empirically. I have done dozens of "runs", with varying specifications, on more than a dozen different arrangements of components. I evaluate these runs by determining why particular alternatives are eliminated, and by examining the "optimal solutions" resulting.

The system appears to eliminate only invalid designs. It frequently surprises me, but I always find either a correctable bug, or that my understanding of the design problem was incomplete.

I am also confident that the designs selected are "optimal" with respect to the cost function, but my confidence here is based on the simplicity and clarity of the optimization process rather than on empirical results. Finding an optimum solution by hand is extremely slow on even these simple design problems, and no optimization program I know of can easily be set up for problems of this kind. Even an exhaustive check of combinations of com-

ponents would still involve sets of operating conditions, hence require most of the mechanisms of my compiler and not constitute an independent check. The most I can say, as a human designer, is that the designs produced look like they could well be optimal.

A subtler question is whether the program eliminates all the implementations it should—whether its rule set is complete enough to guarantee that the designs it produces will work. It is not, in three senses. First, I know that there are propagation operations I have not yet implemented. I implement operations only as needed, because new operations slow the system and require testing. Second, as I discuss later, the compiler does not propagate every specification it could.

Third, and pragmatically most important, the selected design can always be unsatisfactory because of criteria not represented in my component models. My formalism imposes restrictions on the criteria it can represent. In particular, equations must be algebraic, and have three variables, though intermediate variables can be used to break up complex equations. We must be able to solve for each variable, and the resulting functions must be continuous and monotonic. The equations must be "instantaneously true"; they cannot values which occur at different times. Values must be non-negative. Specifications must be stated as equations, cost expressions to be minimized, or "hard-edged" intervals. Finally, variables must be divided into only two "causal categories"—parameters, which are fixed at manufacturing, and state variables, which change during operation.

These restrictions limit expressive power. Lack of differential equations probably prevents the system from compiling servo-system designs, or de-

tecting vibration problems. Speed controller catalogs often provide ratios between the highest and lowest controllable speeds, thus relating two different operating conditions. An attempt to model automobile seat design failed because seat-back position is neither a state variable nor a parameter.

Nonetheless, within the domain and problems I have implemented the system appears to select correct designs. It is at present probably less reliable than a very skilled designer working on familiar problems, because very skilled designers make use of information omitted from the catalogs. However, it is probably more reliable, faster, and more likely to produce an optimal design than the average designer.

2.5 Time Complexity

How long does it take to solve these design problems? "About 20 minutes for a problem involving half a million alternatives" is correct but not very useful, since this depends mostly on implementation and hardware. What we really want to know is how the time required to solve the problem increases as the size of the problem increases.

2.5.1 Theoretical Results

I will consider two measures of the size of the problem. The first is the total number of possible alternatives, where an alternative is a combination of catalog numbers without regard to feasibility. This is proportional to C^n, where n is the number of components in the design, and C the average

catalog length for each component.

The program searches for an optimal solution by creating a binary search tree; the forks in the tree are generated by dividing the catalog for a single component into two parts, splitting the "artifact space". The program then pursues the "most promising" daughter design. There is no guarantee that the "most promising" decision will be correct, and unless it is correct most of the time, back-tracking may require time at least proportional to the number of alternatives.

The situation grows even worse when I consider my other measure of size, that is the number of equations involved. The compiler subsumes the conventional constraint propagation of intervals, and it can be shown [7] that the constraint propagation of intervals can run forever. For example, suppose we have two equations, $x = y$ and $x = 2y$, and we start with intervals $0 \leq x \leq 1$ and $0 \leq y \leq 1$. We first conclude from the second equation that $0 \leq y \leq .5$, then from the first that $0 \leq x \leq .5$, then $0 \leq y \leq .25$ and so on. We never arrive (barring round-off error) at the solution, $x = y = 0$.

It may be possible to avoid such pathological cases in real design problems, but even much simpler forms of constraint propagation can require time double-exponential in the number of variables involved, and therefore singly exponential in the number of equations[7].

2.5.2 Empirical Results

Fortunately, my system actually performs much better than the worst case theoretical projections. In figures 2.6 and 2.7, I have used the number of

specifications generated by the searching compiler as my measure of time; this measure is independent of the particular hardware and software implementation. Most of my operations take time proportional to the number of specifications generated. One, the elimination of alternatives, can at worst take time proportional to the number of specifications generated times the average length of the catalogs.

Figure 2.6 shows a semi-logarithmic plot of the number of specifications generated against the number of alternatives. At worst, the number of specifications generated grows according to the logarithm of the number of alternatives.

Figure 2.7 shows a plot of the number of equations involved in the design against the number of specifications generated; growth is no worse than linear with the number of equations. This, in turn, is linear in the number of components in the design.

2.5.3 Explaining the difference

There seem to be five principal reasons why the empirical results are so much better than the worst case predictions.

First, note that eliminating a single catalog number eliminates many alternatives, since that catalog number is involved in a combinatorial set of alternatives.

Second, the artifact space is organized, for example by horsepower. A single specification can eliminate many catalog numbers. More importantly, the optimal solution generally involves the smallest (or nearly the smallest)

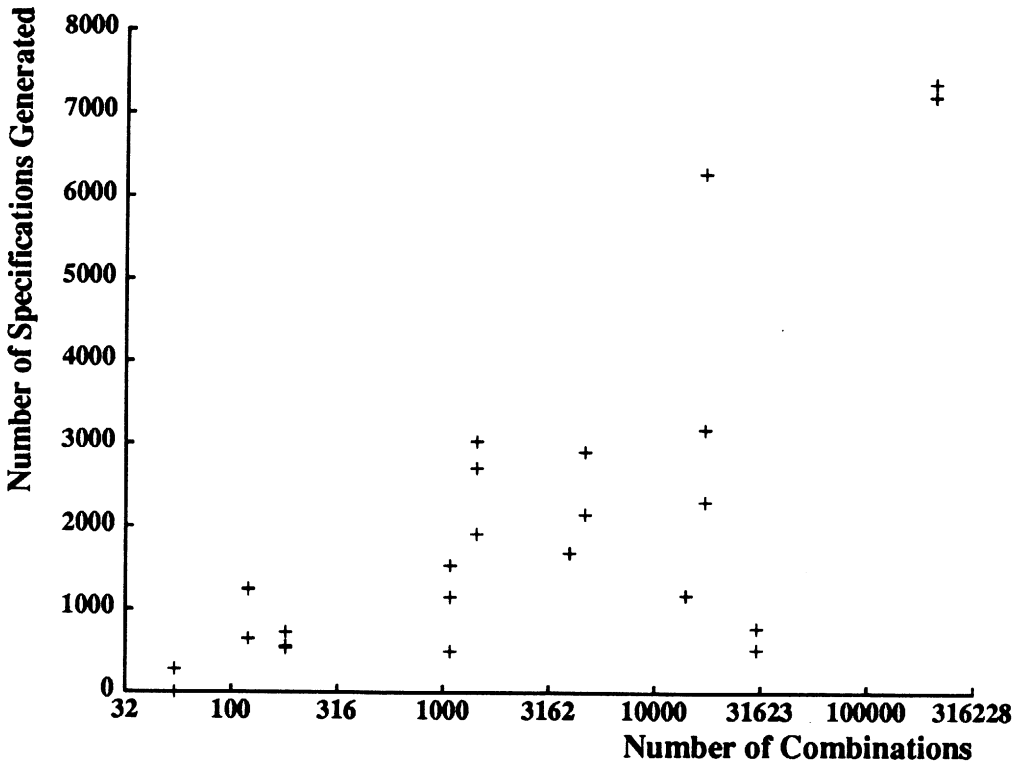


Figure 2.6: Specification generation vs alternatives for a variety of designs

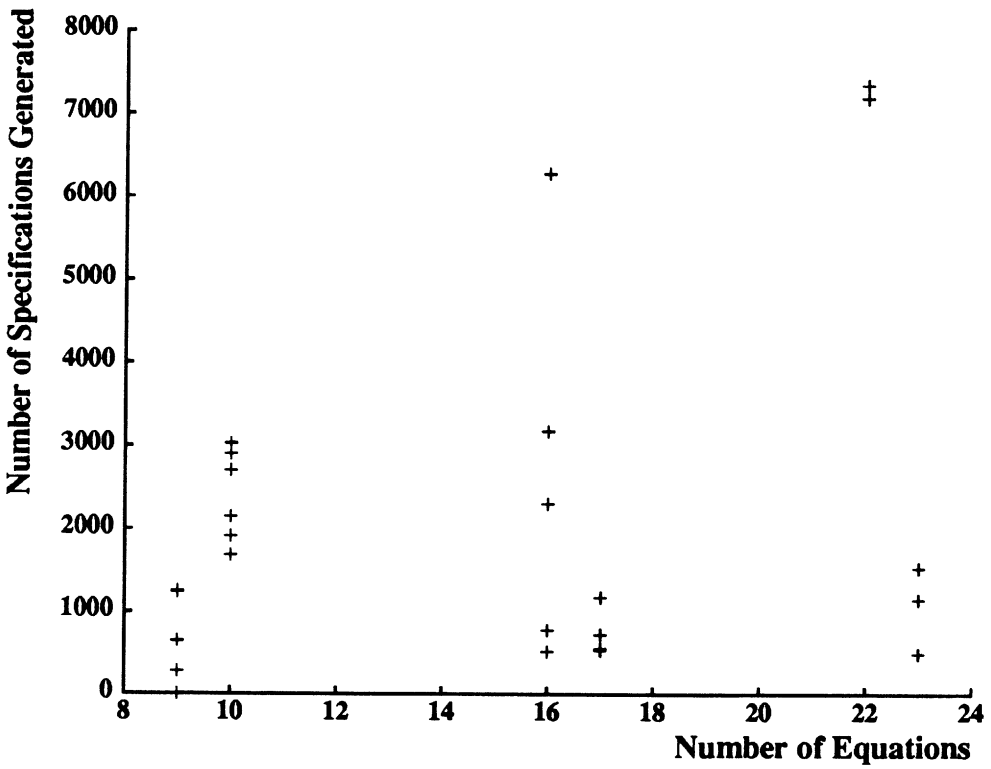


Figure 2.7: Specification generation vs equations for a variety of designs

of the devices meeting the horsepower requirement. Since these are clustered together in the search space, only a few branches of the search tree need be followed.

Third, the equations used to describe mechanical components establish a fairly sparse network between variables. In particular, all information passed between components is channeled into a small number of "port variables", such as rpm and torque. (These components have been selected for manufacturing and cataloging in part because they have relatively simple connections with the rest of a design.) This sparseness helps limit the growth in execution time as a function of the number of equations.

Fourth, some of the propagation operations are correct only if each input specification is independent of the other variables in the equation used. The compiler in fact requires independence for all propagation operations, thus preventing infinite loops of the kind discussed above.

Fifth, I propagate only the "strongest" specifications, for example the tightest required limits.

These last two reasons involve restrictions on the constraint propagation process. We have not proven that these restrictions cannot cause failures to eliminate, but have not observed any such errors in practice.

2.6 Conclusions

To summarize, the compiler has been tested on a range of mechanical and hydraulic power transmission designs; new designs can be entered by the designer in seconds. Results have been correct and optimal for the tested

problems. Time required for solution grows reasonably slowly as the problem grows. These results are evidence of the utility of the theory outlined in this thesis.

Chapter 3

The labeled interval calculus

3.1 Introduction

Suppose that we wanted to design a power train for an ice-cream stirrer (Figure 3.1). I will call this the Toscanini's problem, after a local eatery. Given a range of acceptable stirring speeds, the torques required, and a catalog, we might use the transmission input-output equations $RPM_i = (ratio)(RPM_o)$ and $torque_o = (ratio)(torque_i)$ to systematically eliminate those transmissions unable to provide the required speed with the available motors. Then we might eliminate those motors unable to provide the required torque through any of the remaining transmissions.

I have several observations to make. First, note that in this example we think about *sets* of artifacts (e.g. all Dayton 3-phase motors), rather than particular artifacts (the motor that fell off the loading dock yesterday). Because of manufacturing variation, even a single catalog number designates

a set of different physical artifacts, which may or may not be interchangeable in a particular design. We must also consider sets of operating conditions; for example, the ice cream maker may be nearly empty, or full of cold Double Dutch Chocolate. We cannot always assume that the maximum load is the only one that matters—some electric motors over-heat unless operated at nearly full load.

Second, torque is a *quantitative* property, normally expressed in terms of real numbers. Our reasoning about torque is therefore also quantitative. However, while the torque at a particular operating condition is normally represented by a real number, the torques required by the stirrer under all ice-cream viscosities and fill levels correspond to an interval of real numbers (say those from 10 to 40 newton-meters.)

Third, the artifact sets are organized, for example by horsepower and motor speed. We can eliminate large sets simultaneously (e.g. all the motors of less than 1 horsepower).

Finally, the only *mathematical* expressions used in the example were al-

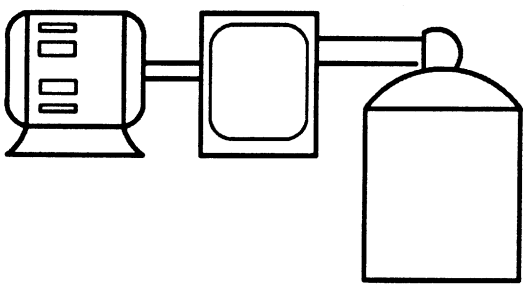


Figure 3.1: The Toscanini's Problem

gebraic equations. Most designers would attack the problem by substituting single values (say for the largest output torque) into the equations, then comparing the results with other single values from catalogs. If asked to justify using calculations on single values to draw conclusions about sets, they would provide intuitive arguments, in English and specific to the particular problem being considered.

We might write these intuitions into an "expert system", and that program might work well in a sufficiently narrow domain. But a compiler should give correct results on every design which can be composed from the schematic elements. It therefore needs a general and precise theory, which can be closely examined and confidently applied to diverse design problems.

3.1.1 Preview

This chapter introduces a theory for quantitative inference about sets of artifacts and operating conditions. The theory provides the basis for a mechanical design compiler which operates by eliminating unsatisfactory alternatives from catalog sets of artifacts.

I will begin with a brief overview of the compiler. I then introduce some operations on real number intervals. From intervals, I build up a language of "labeled-intervals", or "specifications". Then, I illustrate the use of formal operations on this language to perform quantitative inferences in the solution of the Toscanini's problem.

3.2 A design compiler

A user of the compiler creates a design schematic by pointing in sequence at displayed icons. Each icon represents a computational "object", which normally includes a list of catalog numbers. Thus, it also represents the set of real artifacts purchasable by ordering from the catalog. Associated with each catalog number are specifications in the labeled interval language. Other specifications automatically abstracted from these, along with equations built into the schematic object, describe the whole set of artifacts represented by the icon.

The schematic assembly process establishes an identity between corresponding variables for connected components. For example, in the Toscanini's problem the output torque of the transmission is identified with the input torque to the stirrer.

Having assembled a schematic, the user supplies specifications in the labeled interval language for the most convenient objects, usually loads. The objects pass each other these specifications, the specifications abstracted from the artifact sets, and new specifications derived from these by using equations. The objects eliminate from consideration incompatible artifacts (by deleting numbers or groups of numbers from the catalog listing), and abstract new descriptions for the resulting subsets. In the Toscanini's problem, the user might specify the range of torques required at the stirrer input shaft. This information, propagated through equations in the the transmission object, would eliminate those motors unable to supply enough torque to drive the load through *any* of the transmissions under consideration.

Since the information reaching the motor object is about *all* the possible combinations of transmission and load, the compiler does not explicitly enumerate the alternative combinations of motor and transmission. This approach may be contrasted with one in which alternatives are generated, evaluated, and discarded or modified. If we think of the design process as searching a space of artifacts, my approach works by eliminating volumes of the space, while the other evaluates designs at points in the space. At any time during my program's operation, the schematic represents the volume of the artifact space which has not been eliminated.

This approach has several advantages.

- Manufacturing tolerances and operating condition variations are represented explicitly.
- The program need not examine each alternative individually.
- Elimination inferences, unlike choice inferences, can be confidently made from partial information. For example, my program does not yet contain a representation of geometry, but it can still safely eliminate motors providing insufficient torque. It could *not* safely choose a motor—it might not be suitable geometrically.
- The inference system has been designed to produce only statements which are true of *each* of the objects being considered at the present stage of compilation. The sets of artifacts considered at later stages will be subsets of this set, so the statement will still be true of each artifact. Therefore, statements never need to be withdrawn.

- The meaning of design representations is often left intuitive; designs are sometimes said to stand for an "archetype", or a "partially defined object". In contrast, at each stage of the compilation process my representation stands for a well-defined set of physical objects. I can therefore evaluate operations by using physical reasoning about the objects represented before and after a formal operation.

This set-based approach, however, has one significant disadvantage: conventional, single-valued or even "constraint propagating" systems of mathematical inference are inadequate to deal explicitly with sets of artifacts and operating conditions. I now begin building appropriate inference tools based on relationships between variables and intervals of real number values.

3.3 Some Operations on Intervals

We need to work with sets of values, for example the torque required to drive an ice cream stirrer under all load conditions. We might write $0 \leq torque \leq 10$ (in our favorite units), or $torque \in [0\ 10]$ but will instead write $\langle torque\ 0\ 10 \rangle$; for now, the reader can assume these statements mean the same thing.

Using this notation, I will present eight operations on intervals. Because I am trying to convey a general understanding I will present the operations using examples, and claim without proof that under appropriate circumstances the operations are both well defined and computable. For more detail, see chapters 4 and 5.

The first five operations used by my design compiler are straightforward, and are illustrated in the following examples.

- Intersection: $\cap(\langle x\ 1\ 4\rangle,\ \langle x\ 2\ 6\rangle) \longrightarrow \langle x\ 2\ 4\rangle$.
- Not-intersection: $\not\cap(\langle x\ 1\ 4\rangle, \langle x\ 2\ 6\rangle) \longrightarrow$ FALSE.
- Filled-union: $\cup(\langle x\ 1\ 4\rangle, \langle x\ 8\ 10\rangle) \longrightarrow \langle x\ 1\ 10\rangle$.
- Subset: $\subseteq(\langle x\ 10\ 12\rangle,\ \langle x\ 10\ 14\rangle) \longrightarrow$ TRUE.
- Not-subset: $\not\subseteq(\langle x\ 10\ 12\rangle,\ \langle x\ 10\ 14\rangle) \longrightarrow$ FALSE.

I will call the sixth operation RANGE. RANGE takes an implicit equation in three variables and a pair of intervals in two of the variables, and returns the compatible interval in the third variable. More precisely, suppose that $g(x, y, z) = 0$ is the implicit equation, and X and Y are intervals in x and y respectively. Then RANGE$(g, X, Y) \longrightarrow Z$, where Z is the minimal interval such that for every assignment of $x \in X$ and $y \in Y$, there is an assignment of $z \in Z$ which satisfies g.

Let us do an example. Suppose that in the Toscanini's Problem, we had available transmission ratios only from 2 to 4, and we knew that output torques above 8 would damage the stirrer. Figure 3.2 represents the transmission equation, $t_i - \frac{t_o}{ratio} = 0$, by showing lines of constant output torque. From the figure we see that regardless of our choice of transmissions, any motor providing input torque above 4 will induce output torque above 8. We might reasonably conclude that regardless of our choice of transmission we should not use any motor producing running torques above 4. The RANGE

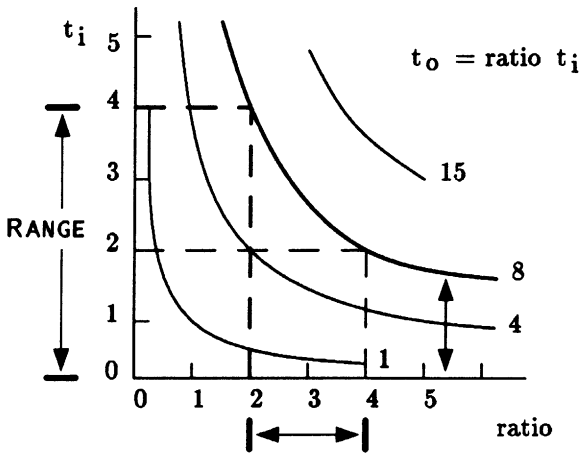


Figure 3.2: An illustration of the RANGE operation

operation produces the appropriate interval:

$$\text{RANGE}(t_i - \frac{t_o}{ratio} = 0,\ \langle t_o\ 0\ 8\rangle,\ \langle ratio\ 2\ 4\rangle) \longrightarrow \langle t_i\ 0\ 4\rangle.$$

The RANGE operation is equivalent to what is usually called the constraint propagation of inequalities, and has been well explored[7]. However, it is not the only operation of interest. Suppose that instead of saying that the stirrer will be damaged by torques above 8, we say that torques ranging from 0 to 8 may be required to drive it. We should conclude that we need motors able to provide torques ranging ranging at least from 0 to 2; see Figure 3.3. I will call the operation producing this interval DOMAIN; it can be defined as an inverse of RANGE. For example,

$$\text{DOMAIN}(t_i - \frac{t_o}{ratio} = 0,\ \langle t_o\ 0\ 8\rangle,\ \langle ratio\ 2\ 4\rangle) \longrightarrow \langle t_i\ 0\ 2\rangle$$

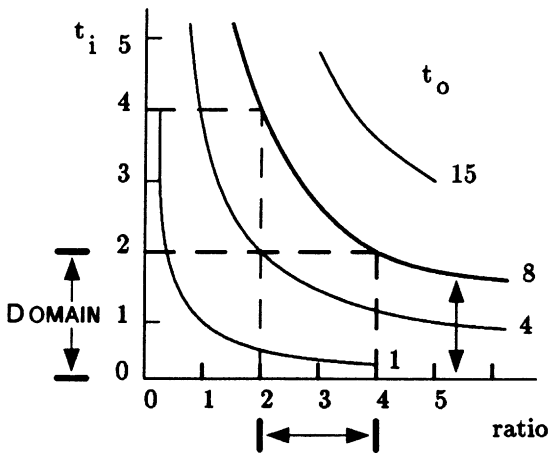


Figure 3.3: An illustration of the DOMAIN operation

precisely because

$$\text{RANGE}(t_i - \frac{t_o}{ratio} = 0,\ \langle t_i\ 0\ 2\rangle,\ \langle ratio\ 2\ 4\rangle) \longrightarrow \langle t_o\ 0\ 8\rangle.$$

Finally, I define the eighth operation, SUFFICIENT-POINTS, as another sort of inverse to RANGE. Suppose in the Toscanini's problem we knew that we had available only motor torques up to 2, and we needed stirrer torques up to 8. Looking at Figure 3.4 we would conclude that any transmission ratio of 4 or above would do. That is,

$$\text{SUFPT}(t_i - \frac{t_o}{ratio} = 0,\ \langle t_o\ 0\ 8\rangle,\ \langle t_i\ 0\ 2\rangle) \longrightarrow \langle ratio\ 4\ \infty\rangle$$

because for all ratios in [4 ∞], the RANGE of the ratio and the input torque includes the output torque. For example, a ratio of 5 would give the output torque interval 0 to 10, which includes the desired interval, 0 to 8.

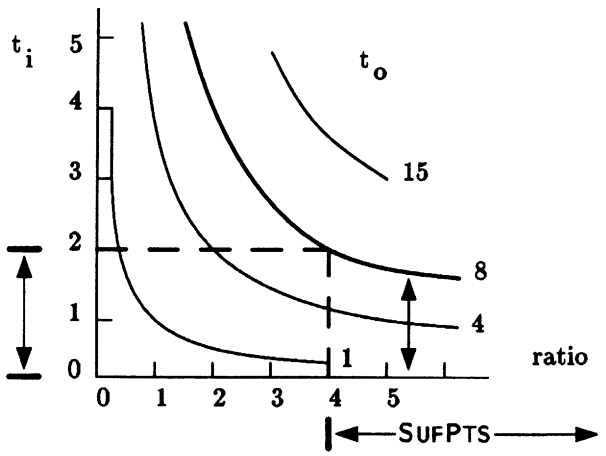


Figure 3.4: An illustration of the SUFFICIENT-POINTS operation

All of the operations presented are sometimes useful in design, but when should we use each one? In these examples we used our experience as designers to decide which operation would produce the desired interval. In a formal system, we need to build the information guiding those decisions into the specifications themselves. I will call these augmented interval statements **labeled intervals.**

3.4 The labeled interval specification language

I will return to the examples of the previous section, but first introduce the language of labeled intervals using an even simpler design problem—selecting one of a set of motors to be connected directly to a load (Figure 3.5).

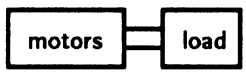


Figure 3.5: A very simple power train

3.4.1 Limits and operating regions

Suppose that we know that each of some set of motors can produce torques throughout the interval 0 to 20, but that damage may result to the load if the torque goes above 10. We want to eliminate these motors from consideration. Given only the intervals ($\langle t\ 0\ 20\rangle$, $\langle t\ 0\ 10\rangle$) a program would not have enough information to specify what operation to use. For example, if the larger interval applied to the load and the smaller to the motors, we would not eliminate the motors. We can attach the information required using the following labels.

The **Limits** label, symbolized by $[^{only}\ ]$, indicates that values of the variable will or must be drawn *only* from the interval. Thus, $\langle [^{only}\ ]\ t\ 0\ 10\rangle$ means that the torque must not reverse or go above 10. Similarly, the tolerance on a bearing inner diameter can be expressed as $\langle [^{only}\ ]\ d_i\ 2.99\ 3.01\rangle$.

The **Operating-Region** label, symbolized by $\overset{every}{\leftrightarrow}$, indicates that the variable will or must assume *every* value in the interval; $\langle \overset{every}{\leftrightarrow}\ t\ 0\ 20\rangle$ indicates that the motor torque can at least range from 0 to 20 (and perhaps beyond.)

I will later define a rule which eliminates these motors because, for the variable t, the operating-region interval is not a subset of the limit interval.

3.4.2 Required, Assured, and No-stronger labels

Suppose that in our motor-load example we want the load speed to be regulated to between 1750 and 1800 rpm. We introduce a **Required** (**R**) interval label, meaning that the statement must be true for proper function. For the load, we can write $\langle \mathbf{R} \overset{only}{[\ ]} \mathit{RPM}\ 1750\ 1800 \rangle$.

Suppose further that some catalog number designates a set of high-slip motors, capable of regulating the speed only well enough to keep it between 1725 and 1800. I introduce the **No-stronger-possible** (**N**) label, and write for the high slip motors $\langle \mathbf{N} \overset{only}{[\ ]} \mathit{RPM}\ 1725\ 1800 \rangle$. By this I mean that we cannot specify any subset of these motors which guarantees stronger limits. (Because of manufacturing variation, some of them probably do guarantee better speed regulation, but we cannot, within the framework given, select these.) I will define a rule which eliminates these motors because the No-stronger-possible limit interval for *RPM* is not a subset of the Required limit interval.

The final label in this class is **Assured** (**A**), indicating that we are sure a particular statement will be true for all the artifacts represented (under appropriate conditions). Thus for our high slip motors, we have also $\langle \mathbf{A} \overset{only}{[\ ]} \mathit{RPM}\ 1725\ 1800 \rangle$.

We have illustrated the Assured, Required, and No-stronger-possible labels only in conjunction with the Limit ($\overset{only}{[\ ]}$) label, but they can be defined comparably in conjunction with the Operating-Range ($\overset{every}{\leftrightarrow}$) label.

Labeled interval descriptions are models of artifact sets, and we can choose the level of abstraction of the model. For example, there is a torque

curve for each motor type, which would allow more accurate prediction of the speed regulation based on the possible torques. If we chose to include the torque curve in our describing equations, we would apply labeled interval specifications to the equation's coefficients.

In addition to the labels defined above, I designate each quantity as either a **state variable** or a **parameter**. Parameters, such as gear ratio, are fixed at manufacture, while state variables like torque may vary during operation.

Each labeled interval pertains only to a specified set of operating conditions such as start-up or normal operating conditions. I will assume "normal operating conditions" throughout this paper.

3.5 Operations on Labeled Intervals

The key activities of my compiler can be specified by three groups of formal operations on labeled intervals: elimination, abstraction, and specification propagation.

3.5.1 Elimination

These operations eliminate artifact sets whose labeled interval specifications conflict with the specifications imposed by the user or by other parts of the design.

I represent these operations using patterns. Suppose that for our motor-load power train we have the same speed regulation requirement as above: $\langle \mathbf{R} \overset{only}{[\ ]} \mathit{RPM}\ 1750\ 1800 \rangle$. We want to eliminate motors with weaker speed

regulation, say $\langle \mathbf{N} \overset{only}{[\ ]} \mathit{RPM}\ 1725\ 1800 \rangle$. These two specifications match the pattern

$$\langle \mathbf{N} \overset{only}{[\ ]} X\ x_l\ x_u \rangle \not\subseteq \langle \mathbf{R} \overset{only}{[\ ]} X\ x_l\ x_u \rangle \longrightarrow \text{eliminate},$$

with X taken to be the RPM and x_l and x_u the lower and upper bounds of the corresponding intervals.

Since the No-stronger-possible specification is not a subset of the Required specification, the program removes the relevant catalog numbers from the associated list.

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} x \ldots \rangle \not\subseteq \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} x \ldots \rangle$
$\langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} x \ldots \rangle \not\cap \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} x \ldots \rangle$
$\langle \mathbf{N} \overset{only}{[\ ]} x \ldots \rangle \not\subseteq \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} x \ldots \rangle$
$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} x \ldots \rangle \not\subseteq \langle \mathbf{N} \overset{every}{\leftrightarrow} x \ldots \rangle$

Table 3.1: Elimination patterns

All my elimination patterns are shown in Table 3.1 (with the arrow and the word "eliminate" omitted for brevity). When the list "(**R A**)" appears in a pattern, it can be matched against either a Required or an Assured statement.

3.5.2 Abstraction

In the Toscanini's problem, we want to evaluate motor alternatives with respect to the set of all transmissions under consideration. Therefore, we need a set of specifications which describe all the transmissions. The program

abstracts these specifications from the previously encoded descriptions of the individual "catalog number" subsets.

The program uses either the *intersection* or the *filled-union* operation to combine the intervals associated with a given variable and pair of labels in each subset. For assured limits it uses the filled-union operation, so for example it combines $\langle \mathbf{A} \overset{only}{[\ ]} RPM 1150\ 1200 \rangle$ and $\langle \mathbf{A} \overset{only}{[\ ]} RPM\ 1750\ 1800 \rangle$ to form $\langle \mathbf{A} \overset{only}{[\ ]} RPM\ 1150\ 1800 \rangle$. There are six types of labeled interval defined by combining the two label sets (Assured, Required, No-stronger-possible) and (Limit, Operating-Region). Table 3.2 shows the operation appropriate for combining each type of labeled interval.

interval type	operation
$\langle \mathbf{A} \overset{every}{\leftrightarrow} \rangle$	$\cap$
$\langle \mathbf{A} \overset{only}{[\ ]} \rangle$	$\cup$
$\langle \mathbf{R} \overset{every}{\leftrightarrow} \rangle$	$\cap$
$\langle \mathbf{R} \overset{only}{[\ ]} \rangle$	$\cup$
$\langle \mathbf{N} \overset{every}{\leftrightarrow} \rangle$	$\cup$
$\langle \mathbf{N} \overset{only}{[\ ]} \rangle$	$\cap$

Table 3.2: Abstraction operations

3.5.3 Propagating Labeled Intervals Using Equations

I turn now to a more complex question: how can we propagate labeled intervals through equations, so that, for example, the torque requirements for the ice cream stirrers can be converted into torque requirements for the motors?

I introduce two operations on labeled intervals and equations.

The first is represented by the following pattern:

$$\langle(\mathbf{R}\ \mathbf{A})\ [\ ]^{only}\ v_1\rangle\ \&\ \langle(\mathbf{R}\ \mathbf{A})\ [\ ]^{only}\ v_2\rangle\ \&\ g(v_1,v_2,v_3)=0$$
$$\longrightarrow\langle(\mathbf{RubA})\ [\ ]^{only}\ v_3\ \textsc{Range}\rangle.$$

The labeled interval patterns to the left of the arrow are matched with potential inputs to the operation, while the pattern to the right of the arrow defines the form of the output. The "$g(v_1, v_2, v_3) = 0$" matches equations linking the two input variables and the output variable. The "**(R A)**" in the input patterns again indicate that the operation is appropriate for either Required or Assured statements. The "**(RubA)**" in the output indicates that the output will be Required unless both inputs are Assured, in which case it will be Assured. Finally, the "RANGE" in the output pattern indicates that the numeric values are to be found by applying the RANGE operation to the input values.

Suppose again that in the Toscanini's Problem, we have available transmission ratios only from 2 to 4, and we know that torques above 10 would damage the stirrer. The specifications match our pattern:

$$\begin{aligned}
\langle\mathbf{R}\ [\ ]^{only}\ t_o\ 0\ 10\rangle &\sim \langle(\mathbf{R}\ \mathbf{A})\ [\ ]^{only}\ v_1\rangle\\
\langle\mathbf{A}\ [\ ]^{only}\ \mathit{ratio}\ 2\ 4\rangle &\sim \langle(\mathbf{R}\ \mathbf{A})\ [\ ]^{only}\ v_2\rangle\\
\tfrac{t_o}{\mathit{ratio}} - t_i = 0 &\sim g(v_1,v_2,v_3)=0.
\end{aligned}$$

This justifies applying the RANGE operation to form $\langle\mathbf{R}\ [\ ]^{only}\ t_i\ 0\ 5\rangle$. The elimination operations will use this new specification to eliminate any motor producing torques above 5.

The second operation is represented by the pattern

$$\langle(\mathbf{R}\ \mathbf{A})\ {}^{every}_{\leftrightarrow}\ s_1\rangle\ \&\ \langle(\mathbf{R}\ \mathbf{A})\ {}^{only}_{[\ ]}\ p_2\rangle\ \&\ g(s_1, p_2, s_3) = 0$$
$$\longrightarrow \langle(\mathbf{Rub}\mathbf{A})\ {}^{every}_{\leftrightarrow}\ s_3\ \textsc{Domain}\rangle.$$

Reading the pattern, we see that the first input must be an operating-region interval and the second a limit. The first input and the output variables must be state variables, while the second input variable must be a parameter. The output interval is formed by applying the DOMAIN operation to the input intervals. The (**Rub**$\mathbf{A}$) rule is applied again. The idea is that if we need a state variable to take on every value in a certain operating-region, and we have some limited choices of parameters in the equation, then the other state variable must take on values over a sufficiently large interval to satisfy the equation with at least one of the parameters available. If we need torques up to 8 to drive the stirrer, we can match the specifications with the pattern:

$$\begin{aligned}
\langle \mathbf{R}\ {}^{every}_{\leftrightarrow}\ t_o\ 0\ 8\rangle \quad &\sim \quad \langle(\mathbf{R}\ \mathbf{A})\ {}^{every}_{\leftrightarrow}\ s_1\rangle \\
\langle \mathbf{A}\ {}^{only}_{[\ ]}\ ratio\ 2\ 4\rangle \quad &\sim \quad \langle(\mathbf{R}\ \mathbf{A})\ {}^{only}_{[\ ]}\ p\rangle \\
\tfrac{t_o}{ratio} - t_i = 0 \quad &\sim \quad g(v_1, v_2, v_3) = 0.
\end{aligned}$$

We therefore apply the DOMAIN operation to form $\langle \mathbf{R}\ {}^{every}_{\leftrightarrow}\ t_i\ 0\ 2\rangle$; the motors are required to supply torques throughout the operating-region from 0 to 2. Note that this specification does not imply that the input torque t_i can never be greater than 2, but rather that all motors considered must be able to supply torques of at least 2. If at some point the transmissions of ratio 4 are eliminated from consideration, a new labeled interval requiring higher t_i will be generated.

Table 3.3 shows all the propagation operations. Symbols representing the associated equations are omitted for brevity. The list "(p s)" may be matched against either a parameter or a state variable. The ↑ and ↓ operations, given intervals in a variable, extend the variable upward to infinity or downward to zero respectively. The $\overset{some}{\cdots}$ label indicates that the variable must take on at least one value in the interval; see Chapter 4 for details.

3.6 Conclusion

What have I done here?

- Provided an explicit and compositional high level language in which designers can define new systems and problems. Formal operations on this language automatically transform high-level descriptions into detailed descriptions.
- Used design descriptions to explicitly represent *sets of artifacts and operating conditions*, rather than a single or "archetypical" artifact under a single operating condition.
- Conducted optimizing search by progressively narrowing volumes of the artifact space, rather than searching from point to point in that space.
- Added the DOMAIN and SUFFICIENT-POINTS operations on intervals to the RANGE constraint-propagating operation.
- Added the *Operating Region* interpretation of intervals to the *Limit* interpretation.

- Divided specification statements into those which are true of all the artifacts represented (Assured), those which must be true of the final design (Required), and those which may or may not be true but which cannot be strengthened (No-stronger-possible).

- Divided variables into "causality classes" (parameters vs state-variables).

These concepts are sufficient to support design compilation over much of the power transmission system domain; chapters 4 and 5 place them on a formal footing, extend them somewhat, and prove the validity of many of the inferences.

$\langle \mathbf{A} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{A} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle \mathbf{A} \overset{every}{\leftrightarrow} s_3 \text{ RANGE} \rangle$

$\langle \mathbf{R} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{R} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow} s_3 \text{ RANGE} \rangle$

$\langle \mathbf{N} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrow} s_3 \text{ RANGE} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} (p_1\ s_1) \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} (p_2\ s_2) \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{only}{[\ ]} (p_3\ s_3) \text{ RANGE} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} (p_2\ s_2) \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{every}{\leftrightarrow} s_3 \text{ DOMAIN} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} s_2 \rangle \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} p_3 \text{ SUFPT} \rangle$

$\langle \mathbf{N} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{only}{[\ ]} p_2 \rangle \longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrow} s_3 \text{ DOMAIN} \rangle$

$\langle \mathbf{N} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathrm{A}\ \mathrm{R}) \overset{only}{[\ ]} (p_2\ s_2) \rangle \longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrow} s_3 \text{ RANGE} \rangle$

$\langle \mathbf{A} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{only}{[\ ]} p_2 \rangle \longrightarrow \langle \mathbf{N} \overset{only}{[\ ]} s_3 \text{ RANGE} \rangle$

$\langle \mathbf{R} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} p_3 \text{ SUFPT} \rangle$

$\langle \mathbf{R} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow} s_3 \text{ DOMAIN} \rangle$

$\langle \mathbf{N} \overset{only}{[\ ]} (p_1\ s_1) \rangle \;\&\; \langle (\mathrm{A}\ \mathrm{R}) \overset{only}{[\ ]} (p_2\ s_2) \rangle \longrightarrow \langle \mathbf{N} \overset{only}{[\ ]} (p_3\ s_3) \text{ DOMAIN} \rangle$

$\langle \mathbf{R} \overset{only}{[\ ]} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{only}{[\ ]} p_2 \rangle \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} s_3 \text{ DOMAIN} \rangle$

$\langle \mathbf{R} \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{only}{[\ ]} p_{@} \rangle \longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow} s_3 \text{ RANGE} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{only}{[\ ]} (p_2\ s_2) \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{some}{\cdots} s_3 \text{ SUFPT} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{every}{\leftrightarrow} s_2 \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{some}{\cdots} s_3 \text{ SUFPT} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{some}{\cdots} s_2 \rangle$

$\longrightarrow \langle\ (\mathbf{RubA}) \overset{every}{\leftrightarrow} s_3\ (\text{DOMAIN}(s_1,s_2) \cap \text{DOMAIN}(\uparrow(s_1),s_2))$

$\cup$

$(\text{DOMAIN}(s_1,s_2) \cap \text{DOMAIN}(\downarrow(s_1),s_2))\ \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{every}{\leftrightarrow} s_1 \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{some}{\cdots} s_2 \rangle$

$\longrightarrow \langle (\mathbf{RubA}) \overset{some}{\cdots} s_3\ (\text{SUFPT}(s_1,s_2) \cap \text{SUFPT}(\uparrow(s_1),s_2))$

$\cup$

$(\text{SUFPT}(s_1,s_2) \cap \text{SUFPT}(\downarrow(s_1),s_2))\ \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) (\overset{only}{[\ ]} \overset{some}{\cdots}) (p_1\ s_1) \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{some}{\cdots} s_2 \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{some}{\cdots} s_3 \text{ RANGE} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) (\overset{only}{[\ ]} \overset{some}{\cdots}) (p_1\ s_1) \rangle \;\&\; \langle (\mathbf{R}\ \mathbf{A}) \overset{some}{\cdots} s_2 \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{only}{[\ ]} p_3 \text{ RANGE} \rangle$

$\langle (\mathbf{R}\ \mathbf{A}) \overset{some}{\cdots} s_1 \rangle \;\&\; \langle \mathbf{N} \overset{only}{[\ ]} p_2 \rangle \longrightarrow \langle (\mathbf{RubA}) \overset{some}{\cdots} s_3 \text{ DOMAIN} \rangle$

Table 3.3: Inference patterns using equations.

Chapter 4

Extending Constraint Propagation

4.1 Introduction

"Constraint propagation" is often thought to be a key element in design [5, 8, 2, 9, 10, 11, 4, 12, 13, 14], hardware debugging [15] and spatial reasoning [7]. Intervals are among the most general constraints propagated; for example, given $y = 2x$ and $1 \leq x \leq 2$, one concludes $2 \leq y \leq 4$. The meaning and validity of this inference seem intuitively clear, and research attention has generally focused on its computational characteristics.

In fact, I show in this chapter that the meaning of these statements and the validity of this inference, as applied to physical objects, require more attention. More precisely, the statement $1 \leq x \leq 2$ can be considered a relationship between a variable name, an interval of values, and the permis-

sible states of the physical object being described. Reasoning about physical objects can involve at least four different kinds of such relationships. Further, the inference shown exemplifies only one of three useful computations on equations and intervals; each of the three performs correct inferences only for appropriate interval-variable relationships.

I begin with an example demonstrating the utility of three kinds of interval propagation, then introduce four "labels" for interval-variable relationships. The bulk of this chapter defines a variety of propagation inferences, and uses basic set theory, predicate calculus, and analysis to prove their correctness. I conclude with an informal discussion of some issues arising in the application of these ideas to the "mechanical design compiler".

4.1.1 An Example

Figure 4.1 shows graphically the governing equation, $t_o = rt_i$, for an ideal variable-speed mechanical transmission; here t_o and t_i are the output and input torques, and r is the continuously variable "transmission ratio". I use this equation to illustrate three different inferences.

Case A: Suppose that the transmission ratio is limited to the interval from 2 to 4, and that if the output torque goes above 8 or falls to less than 1, it will damage the attached load. This seems clear enough: $2 \leq r \leq 4$, and $1 \leq t_o \leq 8$. We want to pick motors which cannot damage the load, and conclude that the input or motor torque must fall in the interval A, from 0.25 to 4; $0.25 \leq t_i \leq 4$. This is the usual notion of interval constraint propagation.

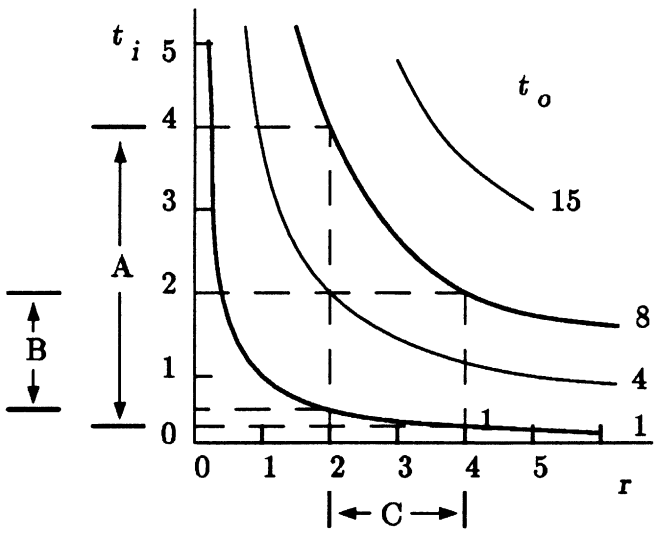


Figure 4.1: Inferences on a Mechanical Transmission

Case B: In contrast, suppose that under the expected operating conditions the output torque must vary throughout the interval from 1 to 8 in order to drive the load. Note that we are not saying that the output torque is limited to the interval from 1 to 8; this interval means something else. With the same limits as in case A on the transmission ratio, we conclude that the motor torque must at least vary over the interval B, that is from 0.5 to 2, or the motor will fail to drive some load. This can't be "interval constraint propagation", since it gives different results on the same equation and intervals.

Case C: Now suppose that the transmission ratio is unknown, that the output torque must vary from 1 to 8 as in case B, and that the input torque is limited to the interval from 0.25 to 4. We conclude that the transmission

must under some operating condition take on at least one value in the interval from 2 to 4, interval C; otherwise, at least one of the required output torques would be unattainable. "Interval constraint propagation" on $0.25 \leq t_i \leq 4$ and $1 \leq t_o \leq 8$ would give the $0.25 \leq r \leq 32$. I will show later that this inference differs from that of Case B as well. Further, the ratio is not limited to the interval from 2 to 4, nor is it required to take on every value in this interval; this interval means something different still from those we have previously encountered.

The transmission equation relates single assignments of values to variables. However, in each case, we used the equation to draw a conclusion about the *set* of values a variable could or should take on. Design is a natural area of application for such reasoning, because the designer must take into account the full variety of conditions under which his design must operate. Mechanical designers are in fact comfortable with the reasoning of the example, but if asked to justify it can provide only intuitive arguments. I will formalize these arguments, beginning by clarifying the possible relationships between variables, the states of an artifact, and intervals of values.

4.1.2 Assignment Intervals, and Equations

Let us suppose ourselves to be discussing an **object** of some sort. We describe this object using a set of **variable** names, and suppose that it can take on various **permissible states**; each state **assigns** each variable a value from the real number line.

We need some notation. I will use S to symbolize the set of permissible

states, and s for an element of S. I will write $X = \langle x\ 0\ 2\rangle$ to mean the set of assignments of values in the interval [0 2] to the variable x; $\mathbf{x} \in X$ to mean such an assignment; and $\mathbf{x}(s)$ to mean the function from states to assignments of the variable x.

Assignments inherit from the real numbers such relations as $<, =$, and infimum, in the obvious way. We can therefore refer to "intervals of assignments". If X is such an interval, then by $\mathbf{x}_l$ I will always mean $\min(X)$ and by $\mathbf{x}_h$, $\max(X)$. I will allow intervals to be infinite, e.g. $\langle x\ 2\ \infty\rangle$, but defer until the last section discussion of the implications of such intervals.

I can now introduce four kinds of statement about objects, their sets of permissible states, and intervals of assignments. I distinguish these by labeling the intervals, as in $\langle label\ x\ x_l\ x_h\rangle$, or just $\langle label\ X\rangle$, and refer to them as **labeled intervals.** (The definitions are repeated in the table at the back of the thesis.)

Definition 4.1 $\langle [\overset{only}{\ }]\ X\rangle \overset{def}{\Longleftrightarrow} \forall s \in S, \exists \mathbf{x} \in X.\mathbf{x}(s) = \mathbf{x}$

That is, the permissible states assign values to x *only* from X; this is the interpretation given all three intervals in case A of the example. This statement is actually a predicate on objects and sets of states, and could be written $\langle [\overset{only}{\ }]\ X\rangle(object, S)$, but as we are considering only a single object and set of states I will leave them implicit.

Definition 4.2 $\langle \overset{every}{\leftrightarrow}\ X\rangle \overset{def}{\Longleftrightarrow} \forall \mathbf{x} \in X, \exists s \in S.\mathbf{x}(s) = \mathbf{x}$

That is, for *every* assignment in X, there exists a permissible state of the object making the assignment. This is the interpretation given both torque intervals in Case B.

Definition 4.3 $\langle \overset{some}{\cdots} X \rangle \overset{def}{\Longleftrightarrow} \exists s \in S. \exists \mathbf{x} \in X. \mathbf{x}(s) = \mathbf{x}$

That is, there exists *some* assignment $\mathbf{x}$ in X and state s in S such that s makes the assignment $\mathbf{x}$. Here we have the interpretation given the transmission ratio interval in case C.

Definition 4.4 $\langle \overset{none}{\bowtie} X \rangle \overset{def}{\Longleftrightarrow} \forall s \in S. \mathbf{x}(s) \notin X$

That is, there is *no* state $s \in S$ such that s makes any assignment in X. As an exception to my normal custom, I here interpret X to be an open interval. If $\langle \overset{none}{\bowtie} X \rangle$ and X is semi-infinite, then we have $\langle \overset{only}{[\ ]} \overline{X} \rangle$, where $\overline{X}$ is the complement of X.

I will interpret equations describing the object as predicates on the permissible states of the the object. More precisely, if the object is described using the equation $G(x, y, z) = 0$, then for every $s \in S$, $G(\mathbf{x}(s), \mathbf{y}(s), \mathbf{z}(s)) = 0$.

I impose tight restrictions on equations, and will discuss in section 4.3 how these restrictions can be accommodated or loosened in practice. First, each equation must be implicit, and in three variables. (If we need equations of more than three variables to describe an object, we can use intermediate variables to convert them into systems of equations.) Second, over the domain of interest the equations must satisfy the **uniqueness** property; that is, if $G(x_0, y_0, z_1) = 0$ and $G(x_0, y_0, z_2) = 0$, then $z_1 = z_2$, and so on for permutations of variable names. Third, the domains of interest must be **compatible**; that is, for any permissible values of x, y there must be a permissible value of z satisfying the equation.

These constraints are sufficient to guarantee that the equation can be solved for each of the three variables, and that the resulting functions are strictly monotonic[1]. Finally, I require that these functions be continuous.

Given $G(x, y, z)$, I will write $g(\mathbf{x}, \mathbf{y})$ to mean the associated function from assignments in x and y to assignments in z.

4.2 Interval Operations and Inferences

I can now formalize three operations on intervals and equations, asking for which permutations of labeled intervals they perform correct inferences.

4.2.1 Conventional Constraint Propagation

I introduce first the operation used in the introduction's Case A.

Definition 4.5 $\text{RANGE}(G, X, Y) = \{\mathbf{z} | \exists \mathbf{x} \in X, \exists \mathbf{y} \in Y. G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

That is, the RANGE of the equation G with respect to the intervals of assignment X, Y is the set of assignments to the variable z such that there exist assignments in X and Y satisfying $G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0$. This is simply the usual image of X, Y under $g(\mathbf{x}, \mathbf{y})$. The continuity of $g(\mathbf{x}, \mathbf{y})$ ensures that

[1] By strictly monotonic, for these functions of two variables, I mean that if $x_1 < x_2$ and $g(x_1, y_1) < g(x_2, y_1)$, then for all $x > x_1$, $g(x, y_0) > g(x_1, y_0)$, and so on for permutations of variable names. It can be shown for these equations that if these inequalities hold, and $y_1 < y_2$ and $g(x_1, y_1) < g(x_1, y_2)$, then for all $x > x_1, y > y_1$, $g(x, y) > g(x_1, y_1)$. Note that the transmission equation is strictly monotonic only over the positive reals.

Z is an interval. Trivially, RANGE is commutative in the intervals; that is, $\text{RANGE}(G, X, Y) = \text{RANGE}(G, Y, X)$.

Recall that by $\mathbf{x}_l$ and $\mathbf{x}_h$ we mean $\min(X)$ and $\max(X)$ respectively; then to compute RANGE we use:

Definition 4.6

$\text{CORNERS}(G, X, Y) = \{g(\mathbf{x}_l, \mathbf{y}_l), g(\mathbf{x}_h, \mathbf{y}_l), g(\mathbf{x}_l, \mathbf{y}_h), g(\mathbf{x}_h, \mathbf{y}_h)\}$.

This leads to

Lemma 4.1 $\text{RANGE}(G, X, Y) = Z$
$= [\min(\text{CORNERS}(G, X, Y))\ \max(\text{CORNERS}(G, X, Y))]$.
Further, if $\mathbf{z}_l = \min(Z) = g(\mathbf{x}_i, \mathbf{y}_1)$, *and* $\mathbf{z}_h = g(\mathbf{x}_j, \mathbf{y}_2)$, *with* $\mathbf{y}_1, \mathbf{y}_2 \in Y$, *then* $\{\mathbf{x}_i, \mathbf{x}_j\} = \{\mathbf{x}_l, \mathbf{x}_h\}$.

The idea is that the maximum and minimum of a monotonic function over a pair of intervals occur at the endpoints of the intervals; further, they occur at different endpoints. The lemma of course holds for permutations of the variable names. The proof follows easily from the monotonicity and continuity of $g(\mathbf{x}, \mathbf{y})$.

For which combinations of labeled intervals does the RANGE operation produce correct inferences? I begin with the most obvious.

Rule 4.1 $\langle [^{only}\ ] X\rangle \& \langle [^{only}\ ] Y\rangle \& G(x, y, z) = 0 \longrightarrow \langle [^{only}\ ] \text{RANGE}(G, X, Y)\rangle$

That is, if for every permissible state $G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0$ is satisfied, $\mathbf{x}(s)$ is in X and $\mathbf{y}(s)$ is in Y, then $\mathbf{z}(s)$ is in the image of X, Y under $g(\mathbf{x}, \mathbf{y})$.

This rule expresses the inference of Case A. Recall that the output torque of the transmission should not go above 8 or below 1; $\langle {}^{only}_{[\ ]}\ t_o\ 1\ 8\rangle$. The transmission ratio could not go below 2 or above 4; $\langle {}^{only}_{[\ ]}\ r\ 2\ 4\rangle$. These, with the equation $t_o = rt_i$, match the antecedents of Rule 4.1. The CORNERS operation substitutes the endpoints of these intervals into $\frac{t_o}{r}$, returning assignments to t_i of $\{0.5, 0.25, 4, 2\}$; and the RANGE operation extracts the maximum and the minimum to form $\langle {}^{only}_{[\ ]}\ t_i\ 0.25\ 4\rangle$, the limits on the input torque.

We also have

Rule 4.2 $\langle {}^{only}_{[\ ]}\ X\rangle \& \langle {}^{some}_{\cdots}\ Y\rangle \& G(x,y,z) = 0 \longrightarrow \langle {}^{some}_{\cdots}\ \text{RANGE}(G,X,Y)\rangle$

Proof: By the definition of ${}^{some}_{\cdots}$, there is some $s \in S$ such that $\mathbf{y}(s) \in Y$, and by the definition of ${}^{only}_{[\ ]}$, $\mathbf{x}(s)$ is certainly in X. Then $\mathbf{z}(s) = g(\mathbf{x}(s), \mathbf{y}(s))$ is in $\text{RANGE}(G,X,Y)$, so $\langle {}^{some}_{\cdots}\ \text{RANGE}(G,X,Y)\rangle$ is satisfied.

In contrast, the possible rule

$$\langle {}^{some}_{\cdots}\ X\rangle \& \langle {}^{some}_{\cdots}\ Y\rangle \& G(x,y,z) = 0 \longrightarrow \langle {}^{some}_{\cdots}\ \text{RANGE}(G,X,Y)\rangle$$

is invalid, because the assignment in X and the assignment in Y need not occur simultaneously. Consider, for example, the labeled intervals $\langle {}^{some}_{\cdots}\ x\ 2\ 3\rangle$, $\langle {}^{some}_{\cdots}\ y\ 1\ 4\rangle$, and equation $xy - z = 0$, and the following consistent and complete set of states:

State	x	y	z
s_1	2.5	0	0
s_2	0	2	0

The rule would incorrectly imply $\langle {}^{some}_{\cdots}\ Z\ 2\ 12\rangle$. The same objection applies to the possible rule, $\langle {}^{some}_{\cdots}\ X\rangle \& \langle {}^{every}_{\leftrightarrow}\ Y\rangle \longrightarrow \langle {}^{some}_{\cdots}\ \text{RANGE}(G,X,Y)\rangle$.

The possible rule

$$\langle {}^{none}\!\bowtie X\rangle \& \langle {}^{none}\!\bowtie Y\rangle \& G(x,y,z) = 0 \longrightarrow \langle {}^{none}\!\bowtie \textsc{Range}(G,X,Y)\rangle$$

is also invalid. Consider an object described only by $\langle {}^{none}\!\bowtie x\ 2\ 3\rangle$, $\langle {}^{none}\!\bowtie y\ 1\ 4\rangle$, $xy - z = 0$. Then a state assigning $x = 1, y = 6, z = 6$ is permissible, and $\langle {}^{none}\!\bowtie Z\ 2\ 12\rangle$ is false. However, let us divide the complement of X into two intervals, $\overline{X}_l = \{\mathbf{x}|\mathbf{x} < \min(X)\}$, and $\overline{X}_h = \{\mathbf{x}|\mathbf{x} > \max(X)\}$. Then, using the symbol $\otimes$ for RANGE, we have

Rule 4.3 $\langle {}^{none}\!\bowtie X\rangle \& \langle {}^{none}\!\bowtie Y\rangle \& G(x,y,z)$
$$\longrightarrow \langle {}^{none}\!\bowtie \overline{\otimes(G,\overline{X}_l,\overline{Y}_l) \cap \otimes(G,\overline{X}_h,\overline{Y}_l) \cap \otimes(G,\overline{X}_l,\overline{Y}) \cap \otimes(G,\overline{X}_h,\overline{Y}_h)}\rangle.$$

The intuition for this rather forbidding expression is that since $\mathbf{x}$ and $\mathbf{y}$ can't be in X and Y, they must be in *overline*X and $\overline{Y}$; these complements can be divided into two intervals each; and $\mathbf{z}$ must be in the RANGE of one pair of such complement intervals. Hence, $\mathbf{z}$ cannot be in the intersection of the complements of those RANGEs. We might suppose that we could label the union of the RANGEs with a $[\,{}^{only}\,]$ label, but in fact that union is not an interval. The intersection of the complements *is* an interval, because: $\overline{X_l}$, $\overline{X_h}$, $\overline{Y}$, and $\overline{Y_h}$ are semi-infinite intervals, the RANGEs of the pairs are also semi-infinite intervals, the complements of the RANGEs are semi-infinite intervals, and the intersection of intervals is an interval.

The formal proof of Rule 4.3 is simple. Let the consequent interval equal Z, and suppose there is some $s \in S$ such that $\mathbf{z}(s) \in Z$. By the antecedents, $\mathbf{x}(s) \in \overline{X}_j$ and $\mathbf{y}(s) \in \overline{Y}_k$, for j and k in $\{h,l\}$. But then $\mathbf{z}(s)$ is in $\textsc{Range}(G,\overline{X}_j,\overline{Y}_k)$, contrary to the definition of Z.

For the final rule of this section, we need:

Definition 4.7 INDEPENDENT(X, Y, S) *if and only if for any* $\mathbf{x} \in X$ *such that* $\mathbf{x} = \mathbf{x}(s_1)$ *and* $\mathbf{y} \in Y$ *such that* $\mathbf{y} = \mathbf{y}(s_2)$, *with* s_1 *and* s_2 *in* S, *then there is an* $s \in S$ *such that* $\mathbf{x}(s) = \mathbf{x}$ *and* $\mathbf{y}(s) = \mathbf{y}$.

As usual, we will often leave S implicit.

If for every pair $\mathbf{x}, \mathbf{y}$ in $X \times Y$ there is a state making these assignments, and G is true in every state, then there is a state making every assignment in $\{\mathbf{z} | \exists \mathbf{x} \in X, \exists \mathbf{y} \in Y . G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$. We therefore have:

Rule 4.4 INDEPENDENT$(X, Y) \& \langle \overset{every}{\leftrightarrow} X \rangle \& \langle \overset{every}{\leftrightarrow} Y \rangle \& G(x, y, z) = 0$
$\longrightarrow \langle \overset{every}{\leftrightarrow} Z$ RANGE$(G, X, Y) \rangle$.

4.2.2 The DOMAIN Operation

I turn now to case B of the introduction, and define a partial inverse of RANGE. That is,

Definition 4.8 DOMAIN$(G, Z, X) = Y \overset{def}{\Longleftrightarrow}$ RANGE$(G, Y, X) = Z$

DOMAIN is partial because for some G, Z, X there is no assignment interval Y satisfying this definition; the computation process given below readily identifies such cases. The following rules apply only when such a Y exists.

Because RANGE is commutative with respect to its interval arguments, DOMAIN$(G, Z, X) = Y$ implies DOMAIN$(G, Z, Y) = X$.

DOMAIN has an equivalent direct definition.

Lemma 4.2 DOMAIN$(G, Z, X) = \{\mathbf{y} | \forall \mathbf{x} \in X, \exists \mathbf{z} \in Z . G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

Proof: Let DOMAIN$(G, Z, X) = Y$, and $Y' = \{\mathbf{y} | \forall \mathbf{x} \in X, \exists \mathbf{z} \in Z.G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$; we must show that $Y = Y'$. Suppose $\mathbf{y}_0 \in Y$. By the compatibility property, for every $\mathbf{x} \in X$, there exists some $\mathbf{z}_0$ such that $G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}_0) = 0$. But by the definition of DOMAIN, RANGE(G,X,Y) = Z, and by the definition of RANGE, $Z = \{\mathbf{z} | \exists \mathbf{x} \in X, \exists \mathbf{y} \in Y.G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$, hence $\mathbf{z}_0$ is in Z; that is, for every $\mathbf{x} \in X$ there is a $\mathbf{z}_0 \in Z$ such that $G(\mathbf{x}, \mathbf{y}_0, z_0) = 0$. $\mathbf{y}_0$ then satisfies $\forall \mathbf{x} \in X, \exists \mathbf{z} \in Z.G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}) = 0$, and $\mathbf{y}_0 \in Y'$.

To show that each point of Y' is in Y, we show first that the endpoints of Y' are in Y. Let $\mathbf{y}'_l = \min(Y')$, and let $g(\mathbf{x}_l, \mathbf{y}'_l) = \mathbf{z}_1$, $g(\mathbf{x}_h, \mathbf{y}'_l) = \mathbf{z}_2$; by the definition of Y', $\mathbf{z}_1$ and $\mathbf{z}_2$ are in Z.

At least one of $\mathbf{z}_1, \mathbf{z}_2$ must be an endpoint of Z. To prove this, we assume the converse, $\mathbf{z}_l < \mathbf{z}_1 < \mathbf{z}_h$, and $\mathbf{z}_l < \mathbf{z}_2 < \mathbf{z}_h$. Since $g(\mathbf{x}, \mathbf{y})$ is continuous and monotonic, we can choose some point $\mathbf{y}' < \mathbf{y}'_l$, but sufficiently close to $\mathbf{y}'_l$ that $\mathbf{z}_l < g(\mathbf{x}_l, \mathbf{y}') < \mathbf{z}_h$ and $\mathbf{z}_l < g(\mathbf{x}_h, \mathbf{y}') < \mathbf{z}_h$. But then, by the monotonicity of $g(x, y)$ with respect to x, $\mathbf{z} = g(\mathbf{x}, \mathbf{y}') \in Z$ for all $\mathbf{x} \in X$, and by the definition of Y', $\mathbf{y}' \in Y$. This contradicts the definition of $\mathbf{y}'_l$ as $\min(Y')$; hence, the assumption is false, and at least one of $\mathbf{z}_1, \mathbf{z}_2$ is an endpoint of Z.

Let $\mathbf{z}_k$ designate the element of $\{\mathbf{z}_1, \mathbf{z}_2\}$ which is an endpoint of Z, and let $\mathbf{x}_j$ designate the corresponding element of $\{\mathbf{x}_l, \mathbf{x}_h\}$; that is, $g(\mathbf{x}_j, \mathbf{y}'_l) = \mathbf{z}_k$. By lemma 4.1, $g(\mathbf{x}_j, \mathbf{y}_m) = \mathbf{z}_k$ for some $\mathbf{y}_m \in \{\mathbf{y}_l, y_h\}$; by the uniqueness property of G, $\mathbf{y}'_l = \mathbf{y}_m$.

We can use symmetrical reasoning to conclude that $\mathbf{y}'_h$ is also an endpoint of Y, then use Lemma 1 again to conclude that they are different endpoints (unless Y is a single point). Y is an interval by definition (RANGEis defined only on interval inputs), so all the assignments between $\mathbf{y}'_l$ and $\mathbf{y}'_h$ are also

in Y, and $Y' = Y$.

To compute DOMAIN, we rely on:

Lemma 4.3
If $\text{DOMAIN}(G, Z, X) = Y$, *then* $\mathbf{y}_l = \min(Y)$ *and* $\mathbf{y}_h = \max(Y)$ *are in* $\text{CORNERS}(g, Z, X)$.

Proof: $\text{RANGE}(G, X, Y) = Z$, so by lemma 4.1 $\mathbf{z}_l$ and $\mathbf{z}_h$ are in $\text{CORNERS}(G, X, Y)$. Thus, for some values i and j in $\{l, h\}$, $g(\mathbf{x}_i, \mathbf{y}_l) = \mathbf{z}_j$. But then by the uniqueness property of G, $g(\mathbf{x}_i, \mathbf{z}_j) = \mathbf{y}_l$, so $\mathbf{y}_l$ is in $\text{CORNERS}(G, Z, X)$. Identical reasoning applies to y_h.

We can therefore compute $\text{DOMAIN}(G, Z, X)$ by generating each possible $Y_t = \langle y\ y_1\ y_2 \rangle$ where $y_1, y_2 \in \text{CORNERS}(G, Z, X)$ and $y_1 \leq y_2$, then testing whether $\text{RANGE}(G, X, Y_t) = Z$.

To formulate our next rule, we need one more definition.

Definition 4.9 *Let* $\mathbf{x}(s_1) < \mathbf{x} < \mathbf{x}(s_2)$ *for some* $s_1, s_2 \in S$; *if and only it this implies that there exists some* $s \in S$ *such that* $\mathbf{x} = \mathbf{x}(s)$, *then* $\mathbf{x}$ *is* STATE-CONTINUOUS.

We then have

Rule 4.5 $\langle \overset{every}{\leftrightarrows} Z \rangle \& \langle [\overset{only}{\ }] X \rangle \& G(x, y, z) = 0 \& \text{STATE-CONTINUOUS}(y)$
$\longrightarrow \langle \overset{every}{\leftrightarrows} \text{DOMAIN}(G, Z, X) \rangle$

Proof: $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ are either positive or negative throughout the domain. There are four permutations of these signs; I consider one in detail. Throughout, let $Y = \text{DOMAIN}(G, Z, X)$. The idea is to show that there must be states making assignments of y on either side of Y.

Suppose $\frac{\partial z}{\partial x} < 0$, and $\frac{\partial z}{\partial y} < 0$. Combining this with the definition of DOMAIN and lemma 4.1 we have $g(\mathbf{x}_h, \mathbf{y}_h) = \mathbf{z}_l$. By the first antecedent of the rule we can choose $s_l \in S$ such that $\mathbf{z}(s_l) = \mathbf{z}_l$. By the second antecedent, $\mathbf{x}(s_l) \in X$, so $\mathbf{x}(s_l) \leq \mathbf{x}_h$. Recall that $G(\mathbf{x}(s), \mathbf{y}(s), \mathbf{z}(s)) = 0$ for all $s \in S$, hence by the uniqueness property of G, $g(\mathbf{x}(s_l), \mathbf{y}(s_l)) = \mathbf{z}(s_l)$. Now, assume that $\mathbf{y}(s_l) < \mathbf{y}_h$; by the assumed signs of the partial derivatives, $\mathbf{z}(s_l) = g(\mathbf{x}(s_l), \mathbf{y}(s_l)) > g(\mathbf{x}(s_l), \mathbf{y}_h) > g(\mathbf{x}_h, \mathbf{y}_h) = \mathbf{z}(s_l)$. The assumption must be invalid, and $\mathbf{y}(s_l) \geq \mathbf{y}_h$.

Symmetrical reasoning leads to the conclusion that there is some $s_h \in S$ such that $\mathbf{y}(s_h) \leq \mathbf{y}_l$. Then, by the STATE-CONTINUOUS assumption, for each $\mathbf{y} \in [\mathbf{y}(s_h)\ \mathbf{y}(s_l)] \supseteq Y$, there is an $s \in S$ such that $\mathbf{y}(s) = \mathbf{y}$.

Rule 4.5 performs the inference of case B. Recall that we kept the limit specification on the ratio, $\langle [\overset{only}{\ }]\ r\ 2\ 4 \rangle$, but changed the output torque specification to require that the output torque take on every value in the interval: $\langle \overset{every}{\leftrightarrow}\ t_o\ 1\ 8 \rangle$. The input torque is STATE-CONTINUOUS. The CORNERS operation again substitutes the endpoints of these intervals into $\frac{t_o}{r}$, returning $\{0.5, 0.25, 4, 2\}$. DOMAIN picks out $\langle \overset{every}{\leftrightarrow}\ t_i\ 0.5\ 2 \rangle$ by substituting into $t_o = r t_i$ and checking the result against $\langle t_o\ 1\ 8 \rangle$; $1 = (.5)(2)$ and $8 = (2)(4)$.

The STATE-CONTINUOUS assumption is physically significant. Suppose for our transmission we had $\langle \overset{every}{\leftrightarrow}\ t_o\ 6\ 12 \rangle$, and $\langle t_i\ 2\ 3 \rangle$. Rule 4.5 gives $\langle \overset{every}{\leftrightarrow}\ r\ 3\ 4 \rangle$, which is correct for our variable speed transmission. However, the requirements could be satisfied with a two-speed geared transmission with ratios 2.9 and 4.1.

Nothing in this section proves the existence of DOMAIN(G, Z, X), and indeed this set may not exist. In Case C, for example, there is no set R of assignments to the ratio r such that RANGE$(t_o - rt_i = 0, \langle t_i\ 0.25\ 4\rangle, R) = \langle t_o\ 1\ 8\rangle$. In this case, however, we can apply the next operation.

4.2.3 The Sufficient-Points Operation

Let us begin by extending RANGE to operate on an interval and a point (of assignment), defining RANGE$(G, Y, \mathbf{x}_0)$ = RANGE$(G, Y, [\mathbf{x}_0\ \mathbf{x}_0])$. Then the SUFFICIENT-POINTS operation is defined by

Definition 4.10 SUFPT$(G, Z, X) = \{\mathbf{y} | Z \subseteq$ RANGE$(G, X, \mathbf{y})\}$

I need to show that if $Y = \{\mathbf{y} | Z \subseteq$ RANGE$(G, X, \mathbf{y})\}$ exists it is an interval; that is, if $\mathbf{y}_1 < \mathbf{y}_2 < \mathbf{y}_3$, and $\mathbf{y}_1, \mathbf{y}_3 \in Y$, then $\mathbf{y}_2 \in Y$. Consider first the case where $\frac{\partial z}{\partial y} < 0$. Since $Z \subseteq$ RANGE$(G, X, \mathbf{y}_1)$ we can find some $\mathbf{x}_i \in X$ such that $g(\mathbf{x}_i, \mathbf{y}_1) \leq \mathbf{z}_l$. Then, $g(\mathbf{x}_i, \mathbf{y}_2) < \mathbf{z}_l$. Alternatively, if $\frac{\partial z}{\partial y} > 0$, find $\mathbf{x}_i$ such that $g(\mathbf{x}_i, \mathbf{y}_3) \leq \mathbf{z}_l$, and again $g(\mathbf{x}_i, \mathbf{y}_2) < \mathbf{z}_l$. By symmetrical arguments there is also some $\mathbf{x}_j \in X$ such that $g(\mathbf{x}_j, \mathbf{y}_2) > \mathbf{z}_h$. Thus, $Z \subseteq$ RANGE$(G, X, \mathbf{y}_2)$, and $\mathbf{y}_2 \in Y$.

There is an equivalent direct definition of SUFPT.

Lemma 4.4 SUFPT$(G, Z, X) = \{\mathbf{y} | \forall \mathbf{z} \in Z, \exists \mathbf{x} \in X. G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

Proof: Let SUFPT$(G, Z, X) = Y$, and $Y' = \{\mathbf{y} | \forall \mathbf{z} \in Z, \exists \mathbf{x} \in X. G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$; we need to show that $Y = Y'$. If $\mathbf{y}_0 \in Y$, let $Z_0 =$ RANGE$(G, X, \mathbf{y}_0) = \{\mathbf{z} | \exists \mathbf{x} \in X. G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}) = 0\}$. But by the definition of Y

and SUFPT, Z_0 is a superset of Z, so certainly $\forall \mathbf{z} \in Z, \exists \mathbf{x} \in X.G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}) = 0$, and $\mathbf{y}_0 \in Y'$. Conversely, if $\mathbf{y}_0 \in Y'$, then $\forall \mathbf{z} \in Z, \exists \mathbf{x} \in X.G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}) = 0$; but then Z is a subset of $\{\mathbf{z} | \exists \mathbf{x} \in X.G(\mathbf{x}, \mathbf{y}_0, \mathbf{z}) = 0\} = \text{RANGE}(G, X, \mathbf{y}_0)$, so $\mathbf{y}_0 \in Y$.

It follows immediately from lemmas 4.4 and 4.2 that $\text{SUFPT}(G, Z, X) = \text{DOMAIN}(G, X, Z)$. This in turn implies that if $\text{SUFPT}(G, Z, X) = Y$, then $\mathbf{y}_l$ and $\mathbf{y}_h$ are in $\text{CORNERS}(G, Z, X)$. Accordingly, as with DOMAIN, we can calculate SUFFICIENT-POINTS by testing various combinations of $\text{CORNERS}(G, Z, X)$.

I consider two inferences using SUFFICIENT-POINTS. First,

Rule 4.6 $\langle \overset{every}{\leftrightarrow} Z \rangle \& \langle \overset{only}{[\ ]} X \rangle \& G(x, y, z) = 0 \& \text{STATE-CONTINUOUS}(y) \longrightarrow \langle \overset{some}{\cdots} \text{SUFPT}(G, Z, X) \rangle$

Proof: Let $\text{SUFPT}(G, Z, X) = Y$, and $\overline{Y}_l = \{\mathbf{y} | \mathbf{y} < \min(Y)\}$. For $\mathbf{y} \in \overline{Y}_l$, at least one endpoint $\mathbf{z}_k$ of Z is such that $G(\mathbf{x}, \mathbf{y}, \mathbf{z}_k) \neq 0$ for any $\mathbf{y} \in \overline{Y}_l, \mathbf{x} \in X$. By the first antecedent, there is an $s_1 \in S$ such that $\mathbf{z}(s_1) = \mathbf{z}_k$, and by the second, $\mathbf{x}(s_1)$ is in X, so $\mathbf{y}(s_1)$ must be greater than $\max(\overline{Y}_l)$; thus, there is an $s_1 \in S$ such that $\mathbf{y}(s_1) \geq \min(Y)$. By a symmetrical argument, there is an $s_2 \in S$ such that $\mathbf{y}(s_2) \leq \max(Y)$. Either at least one of $\mathbf{y}(s_1), \mathbf{y}(s_2)$ is an element of Y, or Y is included in the interval between them, in which case the STATE-CONTINUOUS assumption requires a state for every $\mathbf{y} \in Y$.

This rule performs the inference of Case C. We required the output torque to take on all values in an interval, $\langle \overset{every}{\leftrightarrow} t_o\ 1\ 8 \rangle$, but restricted the input torque, $\langle \overset{only}{[\ ]} t_i\ 0.25\ 4 \rangle$. Rule 4.6 applies since the transmission ratio is continuously variable. CORNERS, using $r = \frac{t_o}{t_i}$ returns $\{4, 32, 0.25, 2\}$. Of these,

RANGE($t_o = rt_i, \langle t_i\ 0.25\ 4\rangle, r = 2$) returns $\langle t_i\ .5\ 8\rangle$, which is a superset of $\langle t_o\ 1\ 8\rangle$. $r = 4$ also passes this test, but not $r = 0.25$ or $r = 32$. Hence the rule requires the transmission ratio to take on at least one value in [2 4]; $\langle \overset{some}{\cdots}\ r\ 2\ 4\rangle$.

For the second inference, we need another predicate on variables.

Definition 4.11 PARAMETER(x) *if and only if there is some single assignment* $\mathbf{x}_0$ *such that for all* $s \in S$, $\mathbf{x}(s) = \mathbf{x}_0$.

In the design context, PARAMETER(x) implies that the value of x is fixed at manufacture.

Rule 4.7 $\langle \overset{every}{\leftrightarrow}\ Z\rangle \& \langle [\overset{only}{\ }]\ X\rangle \& G(x,y,z) = 0 \& \text{PARAMETER}(y) \longrightarrow \langle [\overset{only}{\ }]\ \text{SUFPT}(G,Z,X)\rangle$

To prove this, one applies the same reasoning as for Rule 4.6, then notes that since y takes on only one value, that value must be between $\max(Y)$ and $\min(Y)$.

4.3 Some Application Problems

The larger system of which these rules are a part is described in chapters 2 and 3. Their interpretation must be extended to deal with sets of artifacts, rather than individual artifacts; this is done in Chapter 5. In this section I consider some technical problems arising in applying these rules to a practical system.

4.3.1 Relaxing the Monotonicity Assumption

Most of the equations describing mechanical artifacts are **not** monotonic over the real numbers. However, for a wide variety of designs it is possible to restrict values to the non-negative reals, producing strict monotonicity except perhaps at zero.

The CORNERS function may then involve divisions by zero. We extend division in the obvious ways: divisions of non-zero numbers by zero return ∞; divisions and multiplications of numbers by ∞ return zero and ∞ respectively. On dividing zero by zero, or multiplying zero by ∞, CORNERS returns a list including both zero and ∞.

The DOMAIN operation also needs modification. Consider again the transmission problem, where G is $t_o - rt_i = 0$. Suppose the output torque must assume every value in the operating region $\langle {}^{every}_{\leftrightarrow}\ t_o\ 0\ 8\rangle$, while the input torque is limited by $\langle t_i\ 0\ 2\rangle$. Applying Rule 4.5, the CORNERS operation returns $\{0, \infty, \infty, 0, 4\}$. Now, RANGE$(G, \langle {}^{only}_{[\ ]}\ t_i\ 0\ 2\rangle, \langle r\ 0\ 4\rangle) = \langle T_o\ 0\ 8\rangle$, but in fact there is no need for the transmission ratio to drop to 0; any transmission ratio greater than 4 will do. For this rule, then, we modify the DOMAIN operation so that it looks for the minimal interval in r such that RANGE$(G, T_i, R) \supseteq T_o$. In this case, there is no such interval, and this rule make no inference. Instead, Rule 4.6 returns $\langle {}^{some}_{\cdots}\ r\ 4\ \infty\rangle$.

4.3.2 The Termination Issue

The usual constraint propagation of intervals can fail to terminate, for example[7] on equations $x = y$ and $x = 2y$ starting with intervals $0 \leq x \leq 1$

and $0 \leq y \leq 1$. These equations have the solution $x = y = 0$, but the propagation process approaches this solution asymptotically. It therefore does not allow such infinite loops.

Instead, the program records the variables used in deriving each labeled interval. If an antecedent interval depends on either of the other variables in the antecedent equation, the program does not apply the rule. Termination is thus guaranteed.

Justification of this procedure in the general case is still empirical. However, it is informative to more closely consider conventional constraint propagation, Rule 4.1.

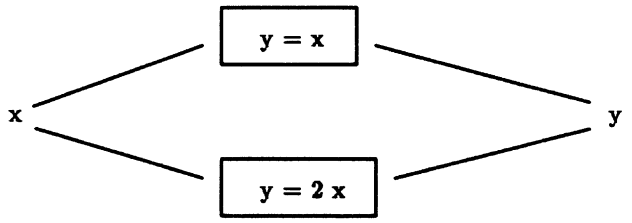


Figure 4.2: A non-terminating propagation net

Figure 4.2 graphically illustrates the non-terminating example. Constraint propagation does not terminate because there are two paths from x to y which, starting with a single value, give different values on reaching y. In the (x, y) plane each equation describes a curve; the equations are both satisfied only at the intersections of the curves. Hence, our convention cutting off propagation after a single cycle leads to looser constraints than are valid.

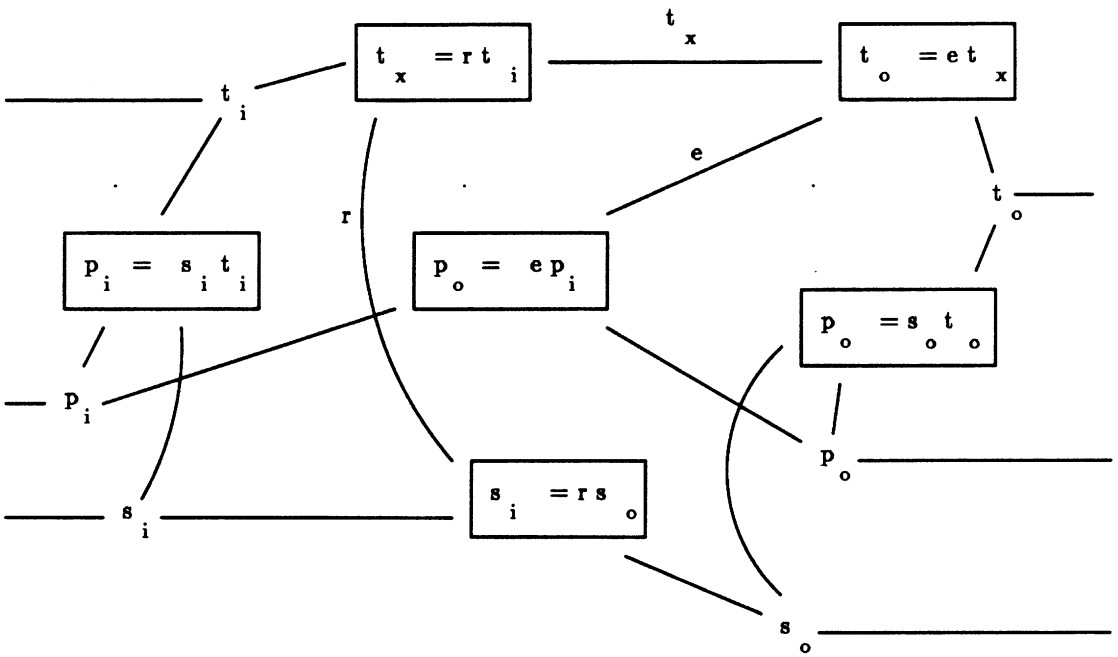


Figure 4.3: The "strongly consistent" mechanical transmission equations

Figure 4.3 shows the equation network modeling a mechanical transmission; in this model we include the power p, and efficiency e. The equation directly relating input and output power is useful; for example, it allows motor power requirements to be calculated without tight bounds on speed or torque. However, as a description of the relationships between single values, it is redundant with the torque and speed equations. More precisely, any pair of equations derived from this network and having all variables in common describes the same surface in n-space. We might refer to this network as **strongly consistent.**

I conjecture that iteration of "constraint propagation" (Rule 4.1) in strongly consistent networks cannot lead to progressive tightening of the limits. To see this, note that such propagation involves taking the maximum and minimum of the CORNERS(o)f the input intervals. Then, CORNERS(o)f the output and one input includes the endpoints of the other input. Hence, moving back and forth along a single propagation chain cannot tighten intervals. But in strongly consistent networks, all the chains between two variables are equivalent in the way they transmit single values. Thus, repeated propagation around loops also cannot progress tighten the intervals.

In this case, strong consistency is required by the physics of transmissions. To date we have been able to describe all of our mechanical artifacts using strongly consistent networks; hence, at least for Rule 4.1, we are justified in cutting off propagation if the intervals are mutually dependent.

4.3.3 Results

I discussed the expressive power of the labeled interval language and the performance of the compiler in detail in chapter 2. Here I remark only that the compiler has been tested on a wide variety of mechanical and hydraulic power train designs, as well a few temperature sensing systems. Some of these designs represent more than a million alternative solutions; the compiler has been able to select a solution, in each case, in less than twenty minutes. The solutions obtained seem consistently optimal; the time required to compile designs seems to grow as the logarithm of the number of alternatives represented, or linearly as the number of equations or variables used to describe them. The compiler has not been used on designs involving feedback loops, or where dynamic (as opposed to quasi-static) performance is important.

More generally, these rules and others, make it possible to reason formally about sets of objects in sets of states. They thus (for some objects) accomplish in a stronger fashion one of the objectives of qualitative physics.

The limitations of the method are in some sense captured by the assumptions of the proofs, though we have seen that some relaxation seems possible in practice. The most critical of these limitations is that the equations be algebraic; we have not yet begun to extend the rules to differential equations.

Chapter 5

Inferences on Sets of Artifacts

5.1 Introduction

The mechanical engineering curriculum is rich in mathematical techniques for describing an existing artifact; indeed, this "applied science" dominates the activities of most departments. On the other hand, designing *new* artifacts is central to engineering practice, but is generally shuffled off to a few projects courses. No pretense is usually made that these courses discuss a science.

A science of design should presumably offer an account of "designs," the evolving descriptions used by designers. One might try to account for designs as words, equations, and lines on paper, without reference to their meaning, but this seems futile. Therefore, a first question is: what do designs mean? Or, what do they stand for?

I claim that design descriptions stand for sets of artifacts[1]. Consider the simple schematics of Figure 5.1. The designer generally chooses between these approaches before knowing precisely what kind of motor or transmission to use; I therefore conclude that if she decides to use a transmission, she bases the decision on what she knows about all the motors and transmissions she might use.

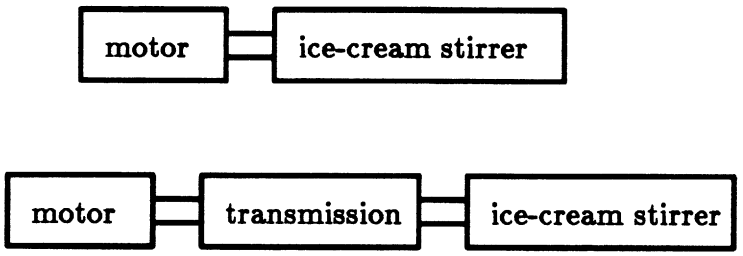


Figure 5.1: A pair of simple mechanical schematics

Consider also the bearing shown in Figure 5.1. Its inner diameter is toleranced to allow for manufacturing variation. The designer must consider these variations in choosing the kind of mating shaft he uses; in fact, he must designate a set of shafts each of which will work in each of the set of bearings designated by the drawing. He reasons about the set of artifacts the drawing represents.

A variety of formalisms have been developed for reasoning about sets of objects. We can use qualitative heuristics to try to capture the expert designer's "feel" for the alternatives; for example it is "usually" better to

[1]This idea was suggested by Chapman's [16] notion of partially completed plans, and Requicha's [17] use of sets to provide a semantics for toleranced drawings.

use a transmission than to apply direct drive. A more rigorous qualitative approach uses "qualitative [18] equations", whose variables take on only three values— zero, minus, and plus.

Unfortunately, the decision whether or not to use a transmission is precisely a quantitative one, depending on a trade-off among such factors as cost, accuracy, and acceleration. We therefore need methods of quantitative reasoning about sets of artifacts.

Probabilistic reasoning is such a method, but seems mis-oriented for some common design problems: what is the probability, for instance, that the designer will use one kind of transmission rather than another? Another approach [19] applies the fuzzy set calculus and a "preference" based semantics to this problem. The precise relationship between these methods and the rules discussed here remains to be determined.

Sets of artifacts can often be described using intervals of real numbers. Thus, the bearings of Figure 5.1 have internal diameters in the interval from 2.99 mm to 3.01 mm. The constraint propagation of intervals has been used in tolerance analysis [9] and in a previous design program of mine [5]. However,

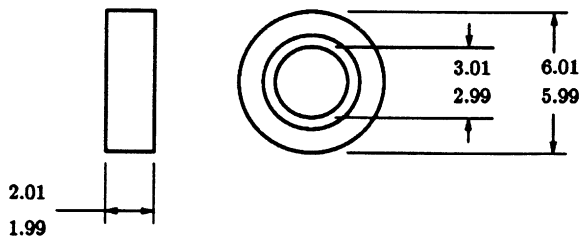


Figure 5.2: "A" ball bearing

as I showed in Chapter 4, the usual idea of interval constraint propagation is but one of several needed to reason about an artifact with takes on various operating conditions. "Labels" on the intervals can be used to determine when these operations should be applied. Here, I extend these ideas further, to account for inferences about sets of artifacts in sets of states.

I will begin by extending the **propagation** rules for single artifacts and sets of states introduced in Chapter 4, to apply to sets of artifacts. I next introduce new rules, which are meaningful only for artifact sets. I then provide rules for **eliminating** basic sets which are certain to be unsatisfactory, and for generating descriptions of "schematic sets" by **abstracting** the descriptions of the included basic sets. Finally, I describe how these operations are interleaved with a form of binary search to "compile" mechanical designs; that is, to automatically transform high level descriptions into detailed descriptions of optimal implementations.

5.2 Labeled Interval Propagation

Artifact sets are built up from **basic sets** of artifacts, those represented by a single catalog number (e.g. "Dayton motor 2N374"). I will designate an individual artifacts a, and basic sets of artifacts A. Individual **schematic symbols** (e.g. for a motor) represent a list of catalog numbers and there associated basic sets; I will represent such catalogs by C. I will often refer to the basic sets $A \in$ C, and sometimes to the artifacts $a \in$ C, which is shorthand for the artifacts in the union of the basic sets in C.

The total schematic of Figure 5.1 then represents the set of all the com-

posite artifacts we might form by combining (in the obvious way) the artifacts represented by the motor, transmission, and load symbols. Connecting the motor symbol to the transmission symbol establishes an identity between appropriate pairs of their describing variables, for example, the motor torque and the transmission input torque; statements about the one are automatically translated into statements about the other.

We cannot choose among the artifacts in a given basic set (this is the defining characteristic of a basic set). In designing, we therefore must choose basic sets such that we can be sure the design will be satisfactory for every combination of the artifacts in these sets.

5.2.1 Extending Single-Artifact Rules to Sets

This motivates the introduction of a pair of labels distinguishing between statements which are known to be true, and statements which must be true in order to have a satisfactory design. Let p be an interval with a single label, say, $\langle [\overset{only}{\ }] \ RPM \ 1725 \ 1880 \rangle$. Then if for some set of motors C_m and normal operating conditions S_n the speed-regulating qualities of every motor in the set will hold the *RPM* in that interval, we can write $\mathbf{A}(\langle [\overset{only}{\ }] \ RPM \ 1725 \ 1880 \rangle, C_m, S_n)$, where the **A** stands for **Assured**. More generally, the statement p is Assured with respect to a set of basic sets C and a set of states S if and only if p is true with respect to S for each artifact in the set.

Definition 5.1 Assured: $\mathbf{A}(p, C, S) \overset{def}{\Longleftrightarrow} \forall a \in C, p(a, S)$

Now consider any of rules 4.1—4.7. If we know that the antecedent statements are true for every artifact in a schematic set C, then the consequent statement is true for every artifact. That is, each rule can be safely modified for use on artifact sets, in such fashion that on Assured inputs it gives an Assured output.

These rules on Assured statements constitute a proof system. If we can formulate the requirements on the design as labeled interval predicate statements, then we should be satisfied with a design if and only if we can prove the truth of the requirement statements from the Assured statements describing the artifacts (and any environmental variables over which we have control). We will use **R**, for **Required**, to label statements which must be true for a satisfactory design. These statements are the goals of the proof system; the Assured statements are axioms.

More formally,

Definition 5.2 Required:

$\mathbf{R}(p, C, S) \stackrel{def}{\Longleftrightarrow} \forall A \in C, \exists a \in A.\neg p(a, S)) \longrightarrow \text{UNSATISFACTORY}(a)$

Note that while Assured statements are true for every artifact represented, we write a Required statement even if only some artifact is each basic set will be unsatisfactory unless the statement is satisfied. As an example, in the second schematic of Figure 5.1 we might know that the set of AC induction motors represented by the motor symbol assure relatively little variation in speed under normal operating conditions S_n; $\mathbf{A}(\langle [\stackrel{only}{\ }] \ RPM \ 1725 \ 1800\rangle, \mathrm{C}_m, S_n)$. We might also know that some transmissions in each of the basic sets represented break down if habitually driven

too fast; $\mathbf{R}(\langle [\overset{only}{\ } \rangle\ RPM_i\ 0\ 1800\rangle, \mathrm{C}_t, S_n)$. Assembling the schematic makes the motor *RPM* and the transmission input *RPM* equivalent, so for every combination of motor and transmission, the first statement proves the second, and the requirement is satisfied.

Now consider again propagation rules 4.1—4.7. Suppose for a particular design and rule, there are Required statements matching the rule's antecedents; for any satisfactory design, the antecedents must be satisfied. But if the antecedents are satisfied, so is the consequent; therefore, in any satisfactory design, the antecedents must be satisfied. For the same reason, if we have one Required antecedent, and one Assured, the consequent is Required.

Propagation of required statements is useful. For example, suppose that the ice-cream stirrer of Figure 5.1 imposes independent requirements on torque and speed; $\langle \mathbf{R} \overset{every}{\leftrightarrow} t\ 0\ 10\rangle$ and $\langle \mathbf{R} \overset{every}{\leftrightarrow} s\ 5\ 10\rangle$. Using Rule 4.4 and the power equation $p = ts$, we conclude $\langle \mathbf{R} \overset{every}{\leftrightarrow} p\ 0\ 100\rangle$ (in appropriate units). The power requirement can be used eliminate under-powered motors before the transmission is selected.

The arguments of the this section allow us to restate Rule 4.1 as the following.

Rule 4.1a: $\mathbf{A}(\langle [\overset{only}{\ } \rangle X\rangle, \mathrm{C}, S)\&\mathbf{A}(\langle [\overset{only}{\ } \rangle Y\rangle, \mathrm{C}, S)\&G(x,y,z) = 0$
$\longrightarrow \mathbf{A}(\langle [\overset{only}{\ } \rangle \textsc{Range}(G, X, Y)\rangle, \mathrm{C}, S)$

Rule 4.1b: $\mathbf{R}(\langle [\overset{only}{\ } \rangle X\rangle, \mathrm{C}, S)\&\mathbf{R}(\langle [\overset{only}{\ } \rangle Y\rangle, \mathrm{C}, S)\&G(x,y,z) = 0$
$\longrightarrow \mathbf{R}(\langle [\overset{only}{\ } \rangle \textsc{Range}(G, X, Y)\rangle, \mathrm{C}, S)$

Rule 4.1c: $\mathbf{A}(\langle [\overset{only}{\ } \rangle X\rangle, \mathrm{C}, S)\&\mathbf{R}(\langle [\overset{only}{\ } \rangle Y\rangle, \mathrm{C}, S)\&G(x,y,z) = 0$
$\longrightarrow \mathbf{R}(\langle [\overset{only}{\ } \rangle \textsc{Range}(G, X, Y)\rangle, \mathrm{C}, S)$

More compactly, we can write

Rule 4.1′:$(\mathbf{R}\ \mathbf{A})(\langle[^{only}]\ X\rangle, \mathrm{C}, S)\&(\mathbf{R}\ \mathbf{A})(\langle[^{only}]\ Y\rangle, \mathrm{C}, S)\&G(x,y,z)=0$
$$\longrightarrow(\mathbf{R}\ \mathbf{A})(\langle[^{only}]\ \textsc{Range}(G,X,Y)\rangle, \mathrm{C}, S)$$

where it is understood that we apply the **A** on the right hand side only if both left sides are **A**. I will refrain from restating all of the rules given in chapter 4; their extensions to sets of artifacts follow the same pattern as for Rule 4.1.

With reference to Rule 4.4, note that the mechanical design compiler has no formal mechanism for establishing the independence of the antecedent statements. Rather, they are assumed independent unless the derivation of either involves another of the variables of the equation, or a dependence is specifically stated by the user. It has been easy for the programmer to correctly employ this convention for Required and Assured statements separately; I suspect it is impossible to correctly employ it for mixed Required and Assured statements. Hence, Rule 4.4 is never applied to mixed **A** and **R** statements.

We have extended rules applying to single artifacts to work on sets of artifacts. In the next section we will consider a new kind of statement which is meaningful only for artifact sets; in the section following, we will derive rules involving this statement.

5.2.2 The "No-stronger" Statement

Let us consider a slightly more complex model of a transmission, one which includes the efficiency e; if p is power, we have $p_o = ep_i$. Suppose that we need an Operating Region of output powers, say $\mathbf{R}(\langle \overset{every}{\leftrightarrows} p_o\ 0\ 8\rangle, \mathrm{C}, S)$. Suppose also that we know that the efficiencies of these transmissions depend on age, lubrication, temperature, speed, and manufacturing variations, but we have in all cases $\mathbf{A}(\langle \overset{only}{[\ ]}\ e\ .5\ .9\rangle, \mathrm{C}, S)$. As design engineers, we would conclude that we should supply enough input power to provide adequate output power at any of these efficiencies; $\mathbf{R}(\langle \overset{every}{\leftrightarrows} p_i\ 0\ 16\rangle, \mathrm{C}, S)$. But we cannot justify this inference on the basis of the specifications given, because we only know from $\langle \mathbf{A} \overset{only}{[\ ]}\ e\ .5\ .9\rangle$ that the efficiency will fall into the interval from .5 to .9. Consistent with this specification, we *might* know that the transmissions are always most efficient at high powers; an equation might show that as the output power approaches 8, the efficiency goes above .8. In this case, we would conclude that in fact input power need only vary from 0 to 10.

Our intuitive designer's reasoning is based on our belief that we will never know a "stronger" statement about the efficiency; by stronger we mean one which confines the efficiency to some part of the interval [.5 .9]. To formalize this idea we define the **No-stronger** label as applied to $\overset{only}{[\ ]}$ statements.

Definition 5.3

$$\mathbf{N}(\langle \overset{only}{[\ ]}\ X\rangle, C, S) \overset{def}{\Longleftrightarrow} \forall A \in C, \forall \mathbf{x} \in X, \exists a \in A. \exists s \in S. \mathbf{x}(a, s) = \mathbf{x}$$

That is, for every basic set in the catalog, for every assignment in X, and for

every distinguishable set of states in S, there is at least one artifact and one state which make the assignment.

We also have a No-stronger statement for Regions of Operation.

Definition 5.4 $\mathbf{N}(\langle \overset{every}{\leftrightarrow} X\rangle, C, S) \overset{def}{\Longleftrightarrow}$

$$\forall A \in C,$$

$$[\exists a \in A. \forall s \in S. \mathbf{x}(a,s) \leq x_h \& \exists a \in A. \forall s \in S. \mathbf{x}(a,s) \geq x_l$$

That is, for every basic set in the catalog, and for every assignment $\mathbf{x}$ not in X, there is at least one artifact whose $\mathbf{x}$ assignments are limited to X.

Some of the rules which follow are valid only if the antecedent statements are **independent** of each other and other the consequent variable. We will need to slightly extend the definition given in Chapter 4 for independence; I will introduce further extensions later.

Definition 5.5 *Suppose there exists some* $a_1 \in A$ *and* $\mathbf{x}_1 \in X$ *such that* $\mathbf{x}_1 = \mathbf{x}(a_1, s_1)$, *and also some* $a_2 \in A$ *and* $\mathbf{y}_1 \in Y$ *such that* $\mathbf{y} = \mathbf{y}(a_2, s_2)$, *with* s_1 *and* s_2 *in* S; *if and only if this implies that there is an* $a \in A$ *and an* $s \in S$ *such that* $\mathbf{x}(a,s) = \mathbf{x}_1$ *and* $\mathbf{y}(a,s) = y_1$, *then* X *and* Y *are independent.*

This is a weakened version of Bayesian independence; instead of saying that knowing the value of one variable tell us nothing about probability of the other assuming a particular value, we are saying roughly that knowing the value of one variable cannot tell us that the probability of the other assuming a particular value is either zero or one.

The compiler *assumes* that specification statements are independent unless they are either specifically labeled as dependent, or their derivation

involves variables which are the same or are dependent. It is up to the programmer modeling the artifact sets to ensure that this assumption is satisfied. I will present informal arguments that this is reasonably possible; formal guidance for programmers, and proof that the assumption can be satisfied consistently without undo weakening of the representation, will have to wait on a better understanding of labeled interval propagation in equation graphs.

5.2.3 The rules

The following rules apply to **N** statements. They are presented with the sets of states and artifacts implicit, and the **N**, **R**, and **A** labels inside the interval statement. Thus, $\langle \mathbf{N} \stackrel{only}{[\ ]} X \rangle$ is here equivalent to $\mathbf{N}(\langle \stackrel{only}{[\ ]} X \rangle, \mathrm{C}, S)$. The rules discussed in section 5.2.1 had been previously proven for single artifacts; the proofs needed only two generic extensions to handle sets of artifacts. In contrast, the following rules are inherently set based, and must be individually proven.

Each of the first set of rules infers a Requirement statement from a No-stronger statement and a Requirement statement. In some of these cases the consequent pertains, not to the set of artifacts described by the antecedents, but to a set which (by virtue of being connected) shares the consequent variable with the set described by the antecedents. We will indicate this fact by showing a C' as an argument to the antecedent. The first rule provides a clarifying example.

Rule 5.1 $\langle \mathbf{R} \overset{every}{\leftrightarrow} Z \rangle$ & $\langle \mathbf{N} \overset{only}{[\ ]} X \rangle$ & PARAMETER(x) & $G(x,y,z) = 0$ $\longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow}$ RANGE$(G,Z,X)(C') \rangle$

The idea is that we need a sufficiently large Operating Region for y to produce the Required Operating Region for z using any $\mathbf{x} \in X$. Consider for example the transmission specifications with which we introduced this section: $\langle \mathbf{R} \overset{every}{\leftrightarrow} p_o\ 0\ 8 \rangle$, $\langle \mathbf{N} \overset{only}{[\ ]} e\ .5\ .9 \rangle$, with equation $t_o - et_i$. Matching them against the pattern, we apply the RANGE operation to get $\langle \mathbf{R} \overset{every}{\leftrightarrow} p_i\ 0\ 16 \rangle$.

However, it is not true that for every transmission in the artifact set there is a permissible state assigning the input power every value in the interval. Rather, in any satisfactory design, for every motor there is a permissible state assigning the input power every value in the interval. This ensures that every motor will be satisfactory with any transmission.

This distinction is not enforced in the current design compiler, which infers the statement about the transmission input power, then translates it into a statement about motor power. This confusion does not appear to have induced errors, and it is possible that for some fundamental reason the distinction does not matter. More likely, it matters in cases I have not yet tested. In any case, the distinction appears to be related to a more fundamental confusion about causality, which I will discuss later.

The proof of Rule 5.1 is straightforward. By the first antecedent, for every artifact a in a satisfactory basic set A, and for every $\mathbf{z} \in Z$, there is a state $s \in S$ such that $\mathbf{z}(a,s) = \mathbf{z}$. By the second antecedent, for every assignment $\mathbf{x} \in X$, every basic set A in the catalog described, and every $s \in S$, there is at least one artifact $a \in A$ such that $\mathbf{x}(a,s) = \mathbf{x}$. Therefore,

by the definition of RANGE, for every $\mathbf{y} \in \text{RANGE}(G, Z, X)$, and every basic set $A \in \text{C}$, there is at least one artifact $a \in A$ and one $s \in S$ such that $\mathbf{y}(a, s) = \mathbf{y}$. But $\mathbf{y}$ also describes a connected basic set of artifacts A', each of which must work with any artifact in the basic set finally selected. We can see that for each such artifact $a' \in A'$ there must be a permissible state $s \in S$ such that $\mathbf{y}(a', s) = \mathbf{y}$. Thus, we have for these connected artifacts $\langle \mathbf{R} \overset{every}{\leftrightarrow} \text{RANGE}(G, Z, X) \rangle$.

Next, we have

Rule 5.2 $\langle \mathbf{R} \overset{only}{[\ ]} Z \rangle \& \langle \mathbf{N} \overset{only}{[\ ]} X \rangle \& \text{INDEPENDENT}(X, Y)$
$\& \text{PARAMETER}(x) \& \text{PARAMETER}(y) \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} \text{DOMAIN}(G, Z, X)(C') \rangle$

Example: Let us use Rule 5.2 to solve a simple "tolerance stacking" problem. Figure 5.2.3 shows a pair of rods, placed end to end; we must choose the tolerance on the first rod in order to guarantee the tolerance on the over-all length. We match the problem against the rule as follows.

$$\begin{aligned} \langle \mathbf{R} \overset{only}{[\ ]} l_t \ 1.98 \ 2.02 \rangle &\sim \langle \mathbf{R} \overset{only}{[\ ]} v_1 \rangle \\ \langle \mathbf{N} \overset{only}{[\ ]} l_2 \ .99 \ 1.01 \rangle &\sim \langle \mathbf{N} \overset{only}{[\ ]} v_2 \rangle \\ l_t - (l_1 + l_2) = 0 &\sim G(v_1, v_2, v_3) = 0. \end{aligned}$$

The rule concludes $\langle \mathbf{R} \overset{only}{[\ ]} l_1 \ .99 \ 1.01 \rangle$.

Proof: The consequent and antecedents of Rule 5.2 pertain to different artifact sets sharing the descriptive variable y. Now suppose the consequent is false; that is, there is some $a \in A$, such that $\mathbf{y}(a, s) = y_0 \notin \text{DOMAIN}(G, Z, X)$, for all $s \in S$. Then, by the definition of DOMAIN, there is some $\mathbf{x}_1 \in X$ such that $g(\mathbf{x}, \mathbf{y}) \notin Z$. From the second antecedent, there is some artifact a_1 such

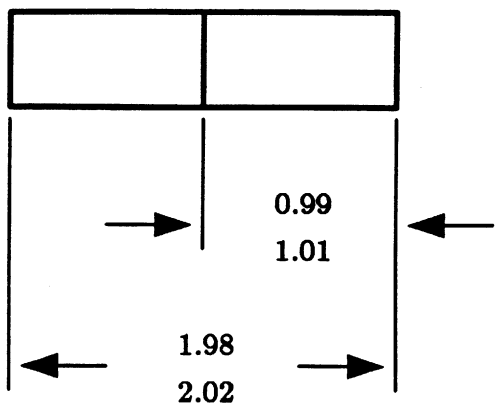


Figure 5.3: Selecting a tolerance

that $\mathbf{x}(a,s) = \mathbf{x}_1$ for all s; by the independence assumption, there is some artifact a_3 assigning both $\mathbf{y}_0$ and $\mathbf{x}_1$. Therefore $\mathbf{z}(a_3,s) = g(\mathbf{y}_0,\mathbf{x}_1)$ is not in Z, violating the requirement of the first antecedent. Hence, for a satisfactory design, there must not be any artifact $a \in A$ and state $s \in S$ such that $\mathbf{y}(a,s) \notin \textsc{Domain}(G,Z,X)$. That is, $\langle \mathbf{R} \overset{only}{[\ ]} \textsc{Domain}(G,Z,X)\rangle$.

Note that we have not in this case required that the consequent and antecedent artifacts be different, although they may be. The assumption of independence is justified by the requirement that x and y be parameters. If they belong to different artifacts, their determining manufacturing process should be independent; if in the same artifact, it is up to the programmer to model any dependence he wishes to take into account.

The next pair of rules are closely related; y is a state variable in one case, a parameter in the other.

Rule 5.3 $\langle \mathbf{R} \overset{every}{\leftrightarrow} Z\rangle \& \langle \mathbf{N} \overset{every}{\leftrightarrow} X\rangle \& \textsc{State-Continuous}(y)$

$$\&G(x,y,z) = 0 \& \textsc{State-Variable}(y) \longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrow} \textsc{Domain}(G,Z,X) \rangle$$

Example: The power equation relates torque, speed, and power: $p = \omega t$, where ω is the angular speed. If we require an operating region of power, say $\langle \mathbf{R} \overset{every}{\leftrightarrow} p\ 2\ 12 \rangle$ and can guarantee only a limited operating region of torques, say $\langle \mathbf{N} \overset{every}{\leftrightarrow} t\ 1\ 2 \rangle$, we can match against the pattern to require a operating region of speeds: $\langle \mathbf{R} \overset{every}{\leftrightarrow} \omega\ 2\ 6 \rangle$.

Proof: Again we can reason by contradiction. $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ are either positive or negative throughout the domain. There are four permutations of these signs; I consider one in detail, since the others involve symmetric reasoning. Throughout, let $Y = \textsc{Domain}(G,Z,X)$. The idea is to show that for a satisfactory design there must be states making assignments to y on either side of Y.

Suppose $\frac{\partial z}{\partial x} < 0$, and $\frac{\partial z}{\partial y} < 0$. Combining this with the definition of DOMAIN and lemma 4.1 we have $g(\mathbf{x}_h, \mathbf{y}_h) = \mathbf{z}_l$. By the first antecedent, for every artifact a in a satisfactory design, there is a state s_l such that $\mathbf{z}(a, s_l) = \mathbf{z}_l$. But by the second antecedent, there is at least one such artifact a_l such that no state s assigns $\mathbf{x}(a_l, s) > \mathbf{x}_h$. Still, by the compatibility property, $g(\mathbf{x}(a_l, s_l), \mathbf{y}(a_l, s_l)) = \mathbf{z}(a_l, s_l) = \mathbf{z}_l$; since $\mathbf{x}(a_l, s) \leq \mathbf{x}_h$, by the assumed signs of the derivatives, $\mathbf{y}(a_l, s_l)$ must be greater than or equal to $\mathbf{y}_h$.

We can use similar reasoning to conclude that there is some $s_h \in S$ and $a_h \in A$ such that $\mathbf{y}(a_h, s_h) \leq \mathbf{y}_l$. But since every artifact in $a' \in A'$ must work with both a_l and a_h, the STATE-CONTINUOUS assumption requires that there is a state in S making every assignment in Y; $\langle \mathbf{R} \overset{every}{\leftrightarrow} Y \rangle$.

For this rule we require neither independence, nor that the antecedents and consequent refer to different components. The key is that the antecedent interval is set by the domain operation; even with the most favorable dependence between parameters, this large an operating region is required.

When y is a parameter, we have

Rule 5.4 $\langle \mathbf{R} \overset{every}{\leftrightarrows} Z\rangle \& \langle \mathbf{N} \overset{every}{\leftrightarrows} X\rangle \& G(x,y,z) = 0$
$\& \textsc{Parameter}(y) \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} \textsc{SufPt}(G,Z,X)(C')\rangle$

Example: We use again the ideal transmission equation: $t_o = rt_i$. If we have $\langle \mathbf{R} \overset{every}{\leftrightarrows} t_o\ 4\ 12\rangle$ and $\langle \mathbf{N} \overset{every}{\leftrightarrows} t_i\ 1\ 6\rangle$, we can match patterns to form $\langle \mathbf{R} \overset{only}{[\ ]} r\ 2\ 4\rangle$. Note that the antecedent input torque requirement really pertains to a different component than the antecedent ratio requirement.

Proof: Let $Y = \textsc{SufPt}(G,Z,X)$, and suppose that there is a satisfactory artifact a_0' such that $\mathbf{y}(a_0', s) \notin Y$, for every s. (Since y is a parameter it does not change with state.) Then by the definition of SufPt, there is a value $\mathbf{z}_0 \in Z$ such that for every $\mathbf{x} \in X$, $g(\mathbf{y}_0, \mathbf{x}) \neq \mathbf{z}_0$; by the first antecedent, for every $a \in A$ there is some $s_0 \in S$ such $\mathbf{z}(a, s_0) = \mathbf{z}_0$. Now, y is shared between C and C′, and every artifact in the finally selected C must work with every artifact in C′, so for every $a \in A$, $\mathbf{y} = \mathbf{y}_0$. By the compatibility property of G, however, for every $a \in A$, there there is some $\mathbf{x}_0 = \mathbf{x}(a, s_0)$, and $\mathbf{z}_0 = g(\mathbf{x}_0, \mathbf{y}_0)$. But this $x_0 \notin X$, which contradicts the second antecedent. Therefore, for a satisfactory design, $y \in Y$; that is, $\langle \mathbf{R} \overset{only}{[\ ]} \textsc{SufPt}(G,Z,X)\rangle$.

There are also rules which prove No-Stronger statements.

Rule 5.5 INDEPENDENT($\langle \mathbf{N} \overset{every}{\leftrightarrow} X \rangle \langle \mathbf{N} \overset{every}{\leftrightarrow} Y \rangle$)$\&G(x,y,z) = 0$
$\longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrow}$ RANGE$(G,X,Y) \rangle$

Example: Suppose we have restricted torque and speed capacity in a speed-controlled motor; $\langle \mathbf{N} \overset{every}{\leftrightarrow} t\ 03 \rangle$, and $\langle \mathbf{N} \overset{every}{\leftrightarrow} \omega\ 04 \rangle$. We apply the rule and the equation $p = t\omega$ to conclude $\langle \mathbf{N} \overset{every}{\leftrightarrow} p\ 012 \rangle$.

To say that the two No-stronger statements are independent is to say that there is a single artifact in every basic set which exhibits both limitations. More formally,

INDEPENDENT($\langle \mathbf{N} \overset{every}{\leftrightarrow} X \rangle, \langle \mathbf{N} \overset{every}{\leftrightarrow} Y \rangle$) $\overset{def}{\Longleftrightarrow} \forall A \in \mathrm{C}, \exists a_0 \in A. \forall s \in S(\mathbf{x}(a_0, s) \in X \& \mathbf{y}(a_0, s) \in Y$

The next rule applies when y is a parameter; that is, when its value is established by the manufacturing process, wear, etc., prior to the imposition of actual loads.

Rule 5.6 INDEPENDENT($\langle \mathbf{N} \overset{every}{\leftrightarrow} X \rangle, \langle \mathbf{N}[\overset{only}{\ }] Y \rangle$)$\&$PARAMETER$(y)$
$\&G(x,y,z) = 0 \longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrow}$ DOMAIN$(G,X,Y) \rangle$

Example: Suppose for example that we can be confident of motor powers, input to our transmission, only up to 16 ($\langle \mathbf{N} \overset{every}{\leftrightarrow} p_i\ 0\ 16 \rangle$), while as before we have only loose bounds on transmission efficiency; $\langle \mathbf{A}\ [\overset{only}{\ }]\ e\ .5\ .9 \rangle$. From the equation $t_o - et_i$ and Rule 5.6 we conclude that we can be sure only of output powers up to 8: $\langle \mathbf{N} \overset{every}{\leftrightarrow} p_o\ 0\ 8 \rangle$. This rule is in some sense the mirror image of Rule 5.1.

Here, we are assuming independence between the antecedent labeled intervals, in the sense that for each assignment $y \in Y$, there is a *set* of artifacts

which assigns y, and there is at least one member of each such set for which **x** is limited to the interval x. The assumption is plausible because y is a parameter. Therefore, if y has not been used in deriving the first antecedent, it seems reasonable that these limitations have been imposed without respect to y.

Proof: For any $\mathbf{y} \in Y$ there is a set of artifacts assigning **y** in every state. There is at least one artifact in each such set for which **x** is limited to X; hence, for any $\mathbf{z} \notin \text{DOMAIN}(G, X, Y)$ there is at least one artifact for which **z** is unattainable.

Finally, we have the following.

Rule 5.7 $\text{INDEPENDENT}(X,Y)\&\langle \mathbf{N} \overset{only}{[\ ]} X\rangle\&\langle \mathbf{A} \overset{every}{\leftrightarrow} Y\rangle\&G(x,y,z) = 0 \longrightarrow \langle \mathbf{N} \overset{only}{[\ ]} \text{RANGE}(G,X,Y)\rangle$

Proof: If there is an artifact in every basic set making each assignment in X, and independently every artifact makes every assignment in Y, then there is an artifact making each assignment is $g(X,Y) = \text{RANGE}(G,X,Y)$.

Justification of the independence assumption: Catalogs for the design compiler have been constructed using a crude notion of causality. That is, **A** and **N** statements are input only in describing an artifact set which is said to "control" the variable; for example, the speed-controlled motor is said to control the speed, and can assert **N**limits on the range of speed. Similarly, an AC induction motor "controls" by its speed regulating characteristics, and can define limits on it ability to regulate. The programmer must ensure the independence of such statements within a single schematic representation; obviously, if this schematic controls a variable, it is independent of variables

in other schematic representations. **A** and **N** statements can be derived only from **A** and **N** statements. Hence, unless they share a derivation, they are independent.

It is precisely this notion of causality which is being challenged by the most recent developments. It now appears that it may be possible to eliminate the distinction between Assured and Required statements, and to replace the No-stronger label with one indicating that a statement is true for at least one artifact in each basic set. That is, instead of writing $\langle \mathbf{N} \stackrel{only}{[\ ]} X \rangle$, we would write $\langle \mathbf{B} \stackrel{every}{\leftrightarrow} X \rangle$, with nearly the same semantics. This change promises to considerably clarify the relationships between rules, while focusing attention on independence relationship.

5.3 Abstraction and Search

This section turns from propagation through equations to address two questions. Given descriptions for basic sets, how can we formulate descriptions of the entire catalog they include? Given such descriptions, how can we select the best basic sets with which to implement a design?

5.3.1 Abstraction

Consider a catalog listing of motors. We can use the printed catalog to formulate labeled interval statements describing the basic sets of motors, as well as equations which assume to apply to all of the motors. However, we need labeled interval statements describing all the motors, so that, for

example, we can draw correct inferences about transmissions before we choose the motor catalog number.

Figure 5.4: Abstracting labeled intervals

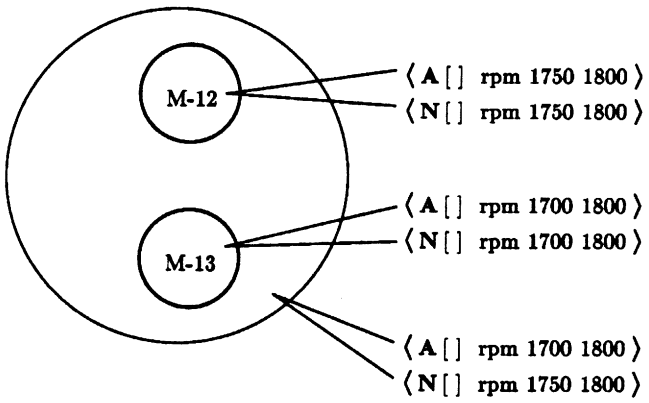


Figure 5.4 shows a Venn diagram for some statements about a pair of basic motor sets, and the super-set which includes them; it illustrates rules 1 and 6 of Table 5.1. Here the symbol $\cup$ stands for the "filled-union", the interval between the minimum and maximum values of the two intervals. Also, if the intersection to be taken is empty, we do not form any new statement. The table explicitly represents artifacts and state sets.

The rules follow trivially from the definitions. For example, in rule 5, every basic set in $C_1 \cup C_2$ clearly has an artifact which is limited to either X_1 or X_2, and hence to their union.

1. $\langle \mathbf{A} \overset{only}{[\ ]} X_1 \rangle(C_1, S) \& \langle \mathbf{A} \overset{only}{[\ ]} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{A} \overset{only}{[\ ]} X_1 \cup X_2 \rangle(C_1 \cup C_2, S)$

2. $\langle \mathbf{R} \overset{only}{[\ ]} X_1 \rangle(C_1, S) \& \langle \mathbf{R} \overset{only}{[\ ]} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{R} \overset{only}{[\ ]} X_1 \cup X_2 \rangle(C_1 \cup C_2, S)$

3. $\langle \mathbf{A} \overset{every}{\leftrightarrows} X_1 \rangle(C_1, S) \& \langle \mathbf{A} \overset{every}{\leftrightarrows} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{A} \overset{every}{\leftrightarrows} X_1 \cap X_2 \rangle(C_1 \cup C_2, S)$

4. $\langle \mathbf{R} \overset{every}{\leftrightarrows} X_1 \rangle(C_1, S) \& \langle \mathbf{R} \overset{every}{\leftrightarrows} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{R} \overset{every}{\leftrightarrows} X_1 \cap X_2 \rangle(C_1 \cup C_2, S)$

5. $\langle \mathbf{N} \overset{every}{\leftrightarrows} X_1 \rangle(C_1, S) \& \langle \mathbf{N} \overset{every}{\leftrightarrows} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{N} \overset{every}{\leftrightarrows} X_1 \cup X_2 \rangle(C_1 \cup C_2, S)$

6. $\langle \mathbf{N} \overset{only}{[\ ]} X_1 \rangle(C_1, S) \& \langle \mathbf{N} \overset{only}{[\ ]} X_2 \rangle(C_2, S) \longrightarrow \langle \mathbf{N} \overset{only}{[\ ]} X_1 \cap X_2 \rangle(C_1 \cup C_2, S)$

Table 5.1: Abstraction rules

5.3.2 Elimination and Search

The set represented by a schematic includes all the artifacts that might be purchased using the catalog numbers associated with the schematic. To select the best implementation, the compiler need only eliminate those catalog numbers which are provably worse than the best. This apparently roundabout procedure has three advantages. First, the descriptions produced by the rules of Table 5.1 are valid for any subset of the catalog described; thus, elimination of any subset leaves the descriptions valid, and the inference pro-

cess is monotonic. Second, elimination processes are local; if some part of the description says that some catalog number is unsatisfactory, the compiler can safely eliminate it regardless of the rest of the description. Third, elimination processes are parallel; we need not be concerned with which catalog numbers we eliminate first. This simplifies programming, and should ultimately allow the compiler to run on parallel hardware.

We can be sure we should eliminate a basic set if some statement describing the basic set conflicts with a some statement describing all the artifacts under consideration. Suppose for example, that we want a motor connected to a load, and we know that for all of the loads under consideration, it is Required that the speed be regulated between 1750 and 1800 rpm; $\langle \mathbf{R} \stackrel{only}{[\ ]} RPM\ 1750\ 1800 \rangle$. If for some basic set of motors we have $\langle \mathbf{N} \stackrel{only}{[\ ]} RPM\ 1725\ 1800 \rangle$, we can eliminate those motors; they cannot provide adequate regulation.

Table 5.2 shows ways in which labeled interval specification statements can conflict. The (**R A**) lists imply that the statement can be either Required or Assured. I omit the quite straightforward proofs.

The elimination process is not guaranteed to leave only a single basic set for each of the components. But for each such incomplete design, the program can create two daughter designs, by splitting some catalog in two. The abstraction process can then be performed on the two sub-catalogs, and propagation and elimination carried out based on the new statements generated. The program can select the most promising daughter design (based on a cost function) on which to repeat the process.

1. $\langle(\mathbf{R}\ \mathbf{A})\ \overset{every}{\leftrightarrows}\ X_1\rangle\&\langle(\mathbf{R}\ \mathbf{A})\ \overset{only}{[\ ]}\ X_2\rangle\&X_1 \not\subseteq X_2$
2. $\langle(\mathbf{R}\ \mathbf{A})\ \overset{only}{[\ ]}\ X_1\rangle\&\langle(\mathbf{R}\ \mathbf{A})\ \overset{only}{[\ ]}\ X_2\rangle X_1 \not\cap X_2$
3. $\langle N\ \overset{only}{[\ ]}\ X_1\rangle\&\langle(\mathbf{R}\ \mathbf{A})\ \overset{only}{[\ ]}\ X_2\rangle\&X_1 \not\subseteq X_2$
4. $\langle(\mathbf{R}\ \mathbf{A})\ \overset{every}{\leftrightarrows}\ X_1\rangle\&\langle N \overset{every}{\leftrightarrows}\ X_2\rangle\&X_1 \not\subseteq X_2$

Table 5.2: Elimination patterns

5.4 Conclusion

This chapter seems likely to be strongly affected by future work. The variety of special definitions of independence are disquieting, as are the number of rules used in the program for which I have not developed proofs. It now appears that it may be possible to replace the No-stronger, Required, and Assured labels with others more directly reflecting the whether a statement is true for every artifact in a basic set, or only some. Such a change should simplify the labeled interval calculus, and make possible more satisfying correctness argument. Questions about independence will remain, to be answered by graph-based arguments. Such developments, however, are beyond the scope of this thesis.

This empty page was substituted for a blank page in the original document.

Chapter 6

The Basic Ideas

6.1 Introduction

In this chapter I step back from the technical details to look at the underlying ideas, summarized in Table 6.1. . By so disengaging the ideas from the particulars of my work I hope to make them useful in other contexts. However, I emphasize that while I have tried to isolate each idea for discussion, they gain much of their power from mutual reinforcement, and from the very details this chapter avoids.

I mention some closely related work, pointing out both similarities (shared ideas) and differences. Where I have misunderstood or omitted such work, I apologize (and would appreciate corrective mail from the researchers).

1. Provide mechanical designers a compositionally-closed design-definition language.
2. Use purchasing or manufacturing procedures to define the set of artifacts represented by a schematic symbol.
3. Automatically transform "high level" descriptions into detailed descriptions of optimal implementations.
4. Use multiple levels of language to link schematic and quantitative descriptions.
5. Use "labeled" intervals to account for operating and manufacturing variation.
6. Automatically abstract schematic-level descriptions from basic-set descriptions.
7. Define a formal operation on intervals and equations which corresponds to the usual idea of constraint propagation; then define its inverses.
8. Formulate rules propagating labeled intervals through equations.
9. Use "ports" to compose designs by equating variables describing connected schematic symbols.
10. Eliminate any distinction between descriptions of "function", "object", "user specifications", and "environment".
11. Eliminate basic sets which can be proven not to work.
12. Search using a cycle of steps which divide or eliminate volumes of the artifact space.

Table 6.1: The Key Ideas

6.2 The Ideas

Idea 1 *Provide to mechanical designers a compositionally-closed design definition language.*

Much design automation work has focused on the control structure of design processes [14, 20, 21, 3]. Such work provides programmers, not designers, tools with which to build programs designing narrowly defined sets of artifacts. The "parametric design" commercial systems also fall into this category.

In contrast, the design compiler provides the designer with a set of primitives for "component types" from which she can in seconds build a description of a design. Such a design, once defined, becomes a building block from which more complex designs can be constructed. This approach allows a single program to automate a wide range of designs. It also tends to enforce rigor; since the language must work for a wide variety of designs, it is hard to hide difficulties with special tricks.

The general notion of compositional descriptive languages is well discussed in [22]. My early efforts to apply it to component selection are discussed in [5, 23]; other mechanical design applications of this idea are described in [4, 24, 8, 2, 10]. Discussion of further ideas will make explicit the differences between these approaches and my own.

Idea 2 *Use purchasing or manufacturing procedures to define the set of artifacts represented by a schematic symbol.*

A list of catalog numbers is associated with the schematic symbol for, say,

a motor. Associated with each catalog number is a manufacturing company which can provide any number of motors of this type. Because of manufacturing tolerances, each of these motors is unique. The schematic symbol for a motor represents the set of all the manufacturable motors for all the catalog numbers. We can, therefore, use set theory and our knowledge of the possible individual motors to rigorously justify inferences on the quantitative description associated with the schematic.

We have used the catalog number, and the purchasing process to which it is an input, to define a set of artifacts. The catalog number *designates* the artifact set; I will later discuss the languages I use to *describe* the set. The catalog number is similar in principle to such designations as "drill-hole, 5mm diameter, 10mm deep"; that is, there is a known process which takes the description and returns any of a predictable set of artifacts.[1]

There are several alternative approaches to defining the meaning of design descriptions. The first and most common is to ignore the problem, assuming that we intuitively know what a design means. A second is to suppose that the design represents an "archetypical" artifact; the notion seems unclear to me.

A more rigorous third approach is to treat the descriptions as defining the sets; given a toleranced description of a hole, we can decide if a given hole belongs in this class [17]. But if we do this we lose the connections among

[1]I have not yet implemented any such designations, and there is a complication. With schematics built from symbols representing sets of cataloged components, we are guaranteed neat hierarchical decompositions; we have no such guarantees for machined components.

the characteristics of a device; in fact, in fact no assurance that it can be manufactured at all.

As a fourth approach one can suppose that a design description represents a particular artifact; the design transformations then change the artifact represented. This approach has been used in "hill-climbing" optimizers for selecting single components [21, 4]; these in turn have been incorporated into larger "expert systems" which in narrowly defined domains can establish the specifications for the individual components [3, 4]. Such non-set-based methods necessarily ignore manufacturing tolerances. Because the artifact represented is continually changing, any inference made about one artifact may have to be re-computed for the next. The search can hang up on local optima, or thrash about making changes in directions orthogonal to that needed. These problems have thus precluded the success of this approach in fully compositional systems.

On the other hand, "design by features" [10, 2] and "variational geometry" [8] systems do provide composable languages suitable for describing single artifacts, not sets. They analyze rather than transform the artifact description; they seem precluded from implementing the following idea.

Idea 3 *Automatically transform "high level" descriptions into detailed descriptions of optimal implementations.*

This embodies the essence of my objective. It is by no means a new idea; rather, it is the essential goal of computer language compilers. I present justifying arguments because I sometimes hear claims that "detailed design" is a solved problem.

A variety of work has been done on "suggestion systems" [24, 25], which provide the user a list of *all* the implementations the program can generate from the inputs, letting her choose between them; by avoiding the hard problem of picking solutions, the researchers have been able to focus on other hard problems (such as the geometric issues I have largely avoided). However, I have several reasons for preferring my own focus.

First, the "detailed design" problem has not been solved. I know of no other tested, compositional, detailed design programs.

Second, the problem is economically important; designers spend much of their time on the detailed, quantitative process of selecting the right components. Programs performing this task would free humans to be "creative". In contrast, suggestion systems may swamp the user with indigestible quantities of poor designs.

Third, detailed design provides a ready test of correctness for ideas which I believe have applications to more "conceptual" or geometric domains. Indeed, the tools I have developed seem to have applications outside design; see chapter 7.

It is sometimes [26] more narrowly argued that design programs should not aim at optimality, for two reasons. First, one can never know that one has the "best" design possible. The utility function used may be wrong. Improvements in materials may always make possible a better design. Second, only for fairly special cases is the "optimality function" smooth enough that one can readily distinguish the global optimum from the local optima.

The first argument is based on correct premises, but misses the point. We

cannot know that we have the best possible design, but we can know that we have the best design which can be produced by a particular program, as measured by the best criteria we can formulate. The second argument is a problem only for some search procedures; we outline one below which is guaranteed to find the globally optimal solution, and empirically appears to do so reasonably quickly.

Idea 4 *Use multiple levels of language to link schematic and quantitative descriptions.*

I actually describe artifact sets using four connected kinds of language; schematic symbols, lists of catalog numbers, networks of equations, and the "labeled intervals" I introduce below. Variables in equations are given meaning by their association with the schematic and the catalog numbers; the variables in turn allow quantitative description of the artifact sets the schematics represent.

In this system the linkage between levels is explicit and direct, as it is in "design by features" [10, 2] and "variational geometry" [8] systems. In contrast, some other programs [25, 26, 4] transform descriptions in one language into descriptions in another using separate operators. These operators can in principal work on more than one composed element; this approach seems to provide a potentially richer toolkit than my own. On the other hand, use of such operators seems to preclude automatic abstraction processes, such as those discussed under Idea 6.

Idea 5 *Use "labeled intervals" to account for operating condition and manufacturing variation.*

There is nothing new in the idea that intervals of real numbers can be used to account for variation. What does appear to be new is the observation that there are many different relationships that can apply between a set of artifacts, a set of operating conditions or states, a descriptive variable, and a set of values for that variable. For example, it is one thing to say that the RPM of a motor is so regulated that under normal operating conditions it never leaves the interval [1000 2000]. It is quite another to say that it is speed controlled, and can be set for any speed in the interval [1000 2000]. These distinctions and more can be captured with a system of "labels"; these examples are labeled by "Limits" and "Region of Operation". Labeled intervals can be given precisely defined meanings using set theory. Useful operations on them can be formally defined, and proven correct.

Idea 6 *Automatically abstract schematic-level descriptions from "basic-set" descriptions.*

If we believe that a schematic should represent a set of artifacts, then we need a description of that set. We could of course write one down in the labeled interval language, but as we shall see, the set represented is continually changing, and writing such descriptions is hard work. But we can, once and for all, write descriptions of the "basic sets" which correspond to individual catalog numbers. We can then automatically generate descriptions for the sets corresponding to a list of catalog numbers. For example, if it is "Required" for some transmissions that the input speed lie in the interval [0 1800] and for others that it lie in the interval [0 3600], then it is Required for all that it lie in the interval [0 3600]; we simply take the union of the

intervals.

Idea 7 *Define a formal operation on intervals and equations which corresponds to the usual idea of constraint propagation; then define its inverses.*

Suppose for example that $z = x + y$, that x is in the interval [1 3], and y is in the interval [2 4]. Then z must be in the interval [3 7]. We can define a purely formal operation (called RANGE) which takes two intervals (with their associated variables) and an equation in three variables, and returns the desired third interval. We can define an inverse to RANGE called DOMAIN; given this equation, the z interval [3 7], and the y interval [2 4], DOMAIN returns the x interval [2 4]. SUFFICIENT-POINTS is another sort of inverse operation; all three of these operations are needed in performing design inferences.

The "constraint propagation of intervals", equivalent to RANGE, has been used for tolerance analysis [9].

Idea 8 *Formulate rules propagating labeled intervals through equations.*

Suppose for a hydraulic pump we are Assured that the input power has an Operating Region (will take on every value) in the interval 100 to 1000 watts, and that the efficiency is Limited is to the interval .8 to 1. We have a rule that if we are Assured that x has a Operating Region X, y has Assured Limits Y, and there is an equation G relating x, y, and z, then we can conclude that z is Assured to have the Operating Region formed by applying the *Domain* operation to G, X, Y.

In this case the equation of interest says that power out equals efficiency times power in, and we conclude an Assured Operating Region for the output power of 100 to 800 watts. Note that "constraint propagation" would give the interval 80 to 1000; we needed the inverse operation.

Most mechanical designers could describe and justify this calculation of the Assured output power Operating Region in intuitive terms. The bulk of this work consists in developing a compact notation for such inference rules; establishing their connections with set theory, logic, and analysis; testing their application to real design problems; and proving their correctness.

I have discussed how some characteristics of artifacts can be represented using labeled intervals, others using equations; how labeled intervals can be propagated through equations, and how descriptions of "schematic sets" can be formulated from their subsets. I turn now to the question of how schematics can be composed, and optimal implementations found.

Idea 9 *Use "ports" to compose designs by equating variables which describe the connected schematic symbols.*

This is a fairly standard notion. There are far fewer kinds of interface between commonly used mechanical artifacts, and variables needed to describe those interfaces, than there are internal arrangements of the artifacts, and variables needed to describe the internal arrangements. This is not an accident; such components are offered for sale partly because they offer simple and well-defined connections to other components. We can therefore identify for a domain a limited class of "port" types, for example shaft-to-shaft connections, or hydraulic ports, each with its characteristic variables, e.g. torque

and flow. These ports types establish identities between the port variables and the internal variables describing each artifact class; e.g., the torque of the input shaft of a transmission, and the input torque of the transmission itself.

When the user tries to connect the motor and transmission schematics, the system can check port types and directions to get the correct connection, then use the port variables to establish an identity between the motor output torque and the transmission input torque. Thus, any statement about one immediately generates a corresponding statement about the other. This uniform propagation mechanism prompts the next idea.

Idea 10 *Eliminate any distinction between descriptions of "function", "object", "user specifications", and "environment".*

In my design compiler, a specification is an equation, a labeled interval, or a schematic linkage. Statements of exactly the same form can be used to describe the "function" of an object (say the output speed of a motor), its "structure" (the height of its shaft), and its environment (the height of the transmission shaft). One might suppose that Requirement statements would always originate with the user, but it is not so; a transmission imposes Required Limits on input speed and torque. This use of the same mechanisms to describe the user's intent, the artifacts, and the environment of the artifacts makes compositionality easy. That the system works well without making these distinctions suggests that they may mean less than often supposed.

Idea 11 *Eliminate basic sets which can be proven not to work.*

This idea may seem inverted; why focus on eliminating bad catalog numbers, instead of picking good ones? But there are good reasons for doing it this way.

Consider an operation which could pick an acceptable basic set for some component of a design, given some specifications. The operation would have to consider **all** of the specifications. But in a compositional system, the number and kind of specifications cannot be known in advance; how can we formulate an operation which considers them all?

On the other hand, consider an elimination operation which examines a statement about the environment in which an artifact set is to operate, and a related statement about the artifact set itself. If the statements conflict, the operator eliminates the basic set. For example, we might have a statement propagated through the transmission to the connected motor, indicating that the horsepower was Required to range throughout the Operating Region from 0 to 1 horsepower. We would eliminate a basic set of motors whose description included an Assured statement that the horsepower is limited to the range from 0 to .5.

Such an operator need only consider two statements at a time, regardless of the rest of the design. The elimination process is inherently modular.

It is also parallel; if the program eliminates one basic set for one reason, and another for a different reason, it does not matter which is eliminated first. This frees the programmer and user from thinking about the "flow of control", allowing them to focus instead on physics. It also means that the program should run well on parallel hardware.

Idea 12 *Search using a cycle of steps which divide and eliminate volumes of the artifact space.*

Design is often thought of as searching a "design" space for the right "design"; the axes of this space are variables describing the design. The compiler instead works in a "artifact space", which has an axis for each symbol in the design schematic. We imagine the catalog numbers for the symbol arranged along the axis in some order. The elimination operation "shortens" an axis, shrinking the space.

We can also cut an axis in two, thus forming two new artifact spaces or daughter designs. The process of division generates new specifications. If for a motor-transmission design we divide the motor catalog, we can eliminate in each daughter design those transmissions which will not work with the daughter designs subset of the motors.

We need a cost function for each schematic symbol (often the function is the same for all axes, say price plus weight); we suppose it to be in "smaller is better" form. The cost function is over those variables which are fixed at manufacture, so for each catalog number there is a small interval of possible values of the cost function; for the whole set of artifacts there will be a much larger interval. We divide again the daughter design with the lowest minimum cost, and repeat the process. This division process forms a tree; we continually divide the most promising leaf of the tree.

We are guaranteed to find the "best" solution; the question is, how long will it take? In theory, we might find ourselves constantly jumping from branch to branch; we might have to search some fixed fraction of all the

potential implementations. However, the artifact space is organized, usually by the size of the components. In consequence, actual design compilations have at worst taken time proportional to the logarithm of the number of alternatives; see Chapter 2.

6.3 Conclusion

I should perhaps conclude with the most general idea of all, the notion of organizing the research as shown in Figure 1.3. I keep such mathematical ideas as "sets" and "closure" closely in mind while developing ideas. I recklessly invent new symbolic notation in order to compactly represent these ideas as formal operations. I test these formalisms using programs, and try to pin down their meaning and prove their validity by tying them to traditional mathematics. I advance opportunistically; the program has frequently been a little ahead of the formalism, the formalism well ahead of the proofs. But I keep trying to coerce them into correspondence.

This approach to problems is hardly new. It has in recent years become increasingly useful because symbolic computation makes formal representations so powerful.

Ultimately, these ideas are valuable if they work. They have worked where I have tested them; I continue to elaborate them, and to extend them to new domains.

Chapter 7

The context of the work

7.1 The Past: A Review of the literature

I have been influenced by a number of general ideas. De Kleer[27] argued that much of our knowledge about the physical world is left implicit by classical mechanics. "Constraint propagation" can be traced to Sutherland[28]. "Silicon compilers"[29] suggested that design operations could be regarded as transformations of formal descriptive languages. Chapman[16] argued that "partially completed plans" represent sets of possible plans; I have directly adapted this idea to physical artifacts.

Work using artificial intelligence methods to study mechanical design can be arranged along a spectrum of increasing abstraction from human design activity. At the most abstract point of this spectrum, Fitzhorn and his students are using Turing machine models to establish fundamental conclusions about the design process[30], while Yoshikawa[31] views design descriptions

as topologies on a space similar to my artifact space. Conversely, at the "human model" end, Waldron and Waldron[32], and Ullman and Dietterich[33] study human designers using the methods of the social sciences.

Toward the "human model" end, Shin-Orr[6], Brown[20], and Mittal, Morjaria, and Dym[14], have developed "expert systems" to design multiple-spindle gear drives, air-cylinders, and paper-paths respectively. These programs use hierarchical control, trial solutions and back-tracking. They apply heuristics obtained by studying experts, and appear to give nearly expert performance in narrowly defined domains.

Near the center of the spectrum we might place work focusing on a single strategy. The Dominic series of programs by Dixon and his students[21], implement a modified "hill-climbing" procedure, searching from point to point in the design space. These select single components, but have been incorporated into a larger system which adjusts the parameters and performance of the hill-climber [3]; a similar approach is in [4]. In using this program, a composed design is encoded by the programmer, rather than assembled from schematics. Also by Dixon and his students are a series of works on "design by features"[10]. Features are geometrically oriented entities (corners, bosses). It appears that compatibility between mating features must be maintained by "hard code", and that in general the systems warn if constraints are violated, rather than using constraints to set values. Papalambros[13], and Rinderle[34] are working on a variety of design support issues and tools, spread across the spectrum.

My work belongs with a cluster slightly further toward the more abstract end of the spectrum. Ulrich and Seering [24] use "generate, test, and debug"

schemes to transform differential equations into schematics, and schematics into more specific pictorial representations. The program does not use quantitative methods for elimination or optimization, instead presenting the human designer with a variety of alternatives. Wood and Antonsson[35, 36] have been exploring the use of fuzzy set theory and fuzzy arithmetic in analyzing designs.

A version of the idea that designs represent sets of artifacts appeared in Requicha's[17] theoretical study of geometric tolerancing.

Agogino and Cagan[37] extend formal optimization methods, for example deriving a torsion tube from a torsion bar by dividing the moment integral into two regions and optimizing over them.

Finally, a variety of work at about this level of abstraction uses constraint propagation. Gossard and his students explore "variational geometry", in which systems of equations are tied to geometric descriptions of parts. Much of this work has been directed to issues of computational efficiency, but see Serrano[1] for a system that allows the designer to use schematics in building an equation network for analysis (not compilation) of a mechanical design. Popplestone[2] et al have used an algebraic constraint propagator as part of a very large system with similar goals. Gross[12]proposes a similar system for architectural design. Fleming[9], propagates the geometric tolerances of parts. Steinberg et al[4] have partially integrated top-down refinement (in the sense of progressively dividing a design problem into modules) and constraint propagation with a hill-climbing mechanism.

These constraint propagation systems, and my earlier work[5, 23], propagate equalities only, or else give intervals the *limit* interpretation and prop-

agate them using only equivalents to the *range* operation. However, see Lozano-Pérez et al[38] for a generalization of the *domain* operation, the *pre-image*, used to formulate robot motion plans under uncertainty.

7.2 Future Work

This section discusses work yet to be done, ranging from near term improvements to the program, to applications of fundamental concepts to new fields, to the development of new concepts. I address these tasks in roughly increasing order of difficulty.

7.2.1 The Near Term: Checks and Improvements

The most important task of the near term is to revise the Assured, Required, and No-stronger labeling system, replacing these with one label indicating the the following statement is true for every artifact in the catalog, and one indicating that it is true only for some artifacts in each basic set. Such a replacement abandons the semantics of the Required statement; this seems possible because the compiler should in fact eliminate any design for which the Required statement is not satisfied. It also abandons any effort to maintain a notion of causality; complete examination of this issue will extend into the medium term.

Further tasks fall into two categories: improving efficiency and ensuring correctness.

Improving Efficiency:

The time required by the program increases fairly slowly as the size of the problem grows, but the program is still too slow for comfortable interaction on realistically large problems. I envision a number of steps to speed it up. The most important and certain in effect involves reducing the rule matching process to an array look-up, by encoding the specification labels as integers.

After a division of the artifact space the compiler propagates specifications and performs eliminations for both daughter designs. Some improvement could be gained by propagating specifications only for the most promising of the daughter designs. We also need a systematic study of the appropriate size for the levels in the abstraction hierarchy, of the extent to which the hierarchy should be allowed to hide subordinate elements from the elimination processes, and of the way in which catalogs should be divided.

The compiler propagates thousands of specifications; human designers only a few. There may be good heuristics for selecting particularly powerful specifications to propagate first. These specifications might perform most of the eliminations. Only late in the compilation process would the program propagate other types of specifications. Selection for early propagation might be based on variable name, or interval type, or might key on parts with particular characteristics—for example, high cost. Heuristics might also be used to select the catalog to be divided, a task now performed by the user. This would allow the program to be run in batch mode.

All of these heuristics would effect operation ordering and hence efficiency only; compilation operations should still preserve correctness. Statis-

tical analysis of specification effectiveness seems the right way to find these heuristics; the program might keep such statistics, adjusting its own approach as it discovered correlations.

Insuring correctness

Some of the compiler's propagation rules can be thought of as a proof system; given some "Assured" statements describing a satisfactory design, these should prove that the "Required" conditions are met. Actually, however, the system has been run only the other way, eliminating components which are incompatible with any such proof; hence the possibility, discussed above, of eliminating the Required and Assured distinction. However, it may be possible install a "mode switch" which would allow the program to be run in "proof mode" as well as "elimination mode". The proof mode would be run on completed designs to check for the completeness of the rule system and the catalog descriptions; any design not able to prove the Requirements should have been eliminated.

There are a variety of design problems which the compiler should solve, but on which it has not been tested. For example, I need to connect a slip clutch *after* a transmission, checking whether the compiler correctly reflects the protection of the transmission against over-torque. I have utilized both "power" specifications (on torque or speed) and "accuracy" specifications (on temperature) but I have not tried to simultaneously deal with both in complex environments. Energy-storing components such as springs and hydraulic accumulators require us the use of "slack variables" in ways I have

not yet tried.

I have proven rules I have not tested empirically, and tested empirically rules I have not proven. These sets need to be brought into correspondence.

7.2.2 The Medium Term: Extensions to the System

Parts of the areas discussed here are straightforward, but each involves extension in areas of substantial uncertainty.

Cost-Function Improvements

The current implementation of cost functions always assigns the same weights to variables with the same name, even in different components. It also requires that the cost function by a simple weighted sum monotonic functions of single variables. Correcting these limitations should be straightforward.

A more significant improvement would incorporate Taguchi's [39] insights into the problem of setting specifications. That is, cost functions should reflect both "reserve capacity" (the distance between the known performance of the system and the boundaries of the required performance) and immunity to variation in noises and other inputs.

Graph Proofs

The correctness of many of the compiler's propagation rules depends on the input specifications being "independent", meaning generally that knowledge of the variable in one specification does not imply knowledge of the variable in the other. (This is closely parallel to Bayesian independence). The precise

formulation of independence depends on the specification types. Further, the compiler does not propagate constraints back and forth between two variables; it can be shown that in equation networks which are not "strongly consistent" (Chapter 4), this can lead to errors.

Fortunately, the designs so far tested are strongly consistent, and it has been possible to set up design problems so that the "independence" criteria are satisfied. We need, however, formal statements of the conventions which must be followed to assure satisfaction of these criteria. More precisely, we need ways to show that they are satisfied for the equation network of a single component, and then inductive proofs that the composition of satisfactory networks can only produce a satisfactory network.

All the designs tested thus far have been tree-structured, without cycles in the component connections. It is not clear how difficult it may be to design systems with cycles.

State trees and qualitative inference

The compiler now incorporates a (short) tree of state sets, whose root node is the set of all operating conditions, under which are arranged normal operating conditions, start-up conditions, and shock-loading conditions. Labeled interval specifications apply to sets of operating conditions, specified by naming the nodes in the tree; specifications applying to the root node apply to all of its subordinate nodes.

At present, equation specifications are assumed to apply to all states; they should in fact follow the same convention as labeled interval specifications.

A slightly more complex extension to the compiler would enable users and component-programmers to extend the state-set tree. These extensions of the compiler should allow representation of multiple quadrant operation, that is, negative values for speeds and torques, as well as non-monotonic equations in general. It should also address discrete quantitative variables, such as the transmission ratio of a gear shift transmission.

The compiler uses qualitative versions of abstraction and elimination operations, but does not have a qualitative analog to propagation through equations. An extensible state tree would provide such a qualitative inference mechanism, with very clear semantics: "if in any of states S, equations E and specifications P apply." Experience will show whether that mechanism is adequately powerful; if not, boolean or predicate calculus statements might be added in parallel with equations.

Completing the Rule Set

The propagation rule set is incomplete, in at least two ways. First, the division of descriptive variables into parameters and state variables is inadequate; other types include noise variables and adjustments. These can, perhaps, simply be given a precedence or causality ordering, and the parameter-state-variable distinction in the rules replaced by ordering requirements.

Second, I have examined only about 80 of the approximately 700 permutations of the interval labels; I have been adding new inferences only as I found them to be needed. It is likely that I have found most of the valid inferences, and certain that I have not found them all. Ensuring completeness

will involve systematically examining the possible inference rules, seeking to either find counter-examples or prove their correctness.

The proofs of correctness found thus far are fairly straightforward. In light of this, and of the number of combinations to be checked, it is intriguing to imagine using a mechanical theorem prover to prove or disprove the other possibilities.

New Uses for Equations

The compiler uses only algebraic equations, with a single interpretation: they relate the values assigned to variables by a single state, or operating condition. A number of improvements are may be possible.

Some benefit would result from equations relating different operating conditions. For example, speed controllers for DC motors are often characterized by the ratio of the maximum to the minimum controllable speed. It is not clear how such equations relate to the propagation rules, whose proofs are based on the "single state" assumption.

More important, abstraction operations now only involve labeled interval specifications, but there are two different ways equations could be used in abstraction. These uses should not violate the "single state" assumption; incorporating them should therefore be fairly straightforward.

First, equations can be used to characterize classes of artifacts, for example, by the power to weight ratio for motors. This ratio could then be used to eliminate AC induction motors (leaving universal and DC motors) in weight-critical applications.

Second, levels of modeling detail can be established, so that simplified equations can be used early in the compilation process, and more complex ones as the number of alternatives considered decreases. This idea is particularly interesting when applied to the abstraction of multiple component sub-designs. The user can now give a linked set of components (say, a motor-transmission pair) a name, and use them in further designs, but once the component group has been connected to other components it loses its identity; the propagation process proceeds as if the entire design had been constructed from scratch using the individual components. I would like to automatically generate new, simplified equations linking the inputs and outputs for such "composite components". This would allow automatic choice between, say, motor-transmission pairs, direct electric drive, and hydraulic pump-motor systems; only after the choice had been made based on the top-level abstraction would expansion of the sub-graph take place.

I conjecture that much of what is often called "conceptual design" simply refers to design with more abstract models. The abstraction mechanism provides a principled way to formulate such models; once formulated, they can be manipulated in the same way as the detailed models on which the system has been tested.

Differential Equations

The compiler uses only algebraic equations, restricting it to quasi-static analysis. Extension to differential equations might follow several paths. For some kinds of analysis, differential equations in the time domain can be trans-

formed into algebraic equations in the frequency domain. The propagation rules may work perfectly on such equations. Other kinds of analysis can be performed using "discrete time steps" and hence algebraic equations to approximate the differential equations. Finally, it may be possible to apply the existing or newly developed inference rules directly to the differential equations themselves.

Geometry

The compiler has thus far addressed only trivial geometric issues. We would like to design systems in which geometry is a geometry is a critical element, and in which specially machined components are involved. One approach would involve the definition of a basic (and large) set of machine elements such as wedges, screws, rotating cams, linkage pairs of varying sorts, motors, brackets, spacers and mounting plates from which complex machines can be constructed. Many of these elements might themselves be composite, with their own internal structure. The critical problem is probably controlling the "port" structure, so as to maintain distinctions and establish appropriate links between the elements while allowing multiple elements to be formed from a continuous piece of material.

The first efforts along these lines may involve the specification of mounting plates for power trains of the sort already tested. Another approach might be toy or kit designs, using Legos or an equivalent. At the next level of difficulty lies the design of relatively decoupled and standardized systems involving multiple machined components, probably in the domain of special production

machinery. Design cost is important in such machines, and the ability to rapidly design them, and to accurately predict their cost and performance, might substantially improve industrial competitiveness.

Larger gains might accrue from standardizing the components from which such machines are constructed. The current approach to "flexible automation" is to build programmability into the production hardware, at considerable cost. It may be more effective for medium scale production to use and re-use standard component modules, automatically generating new mounting plates and brackets to orient them for new tasks. Design compilers can support the rapid and reliable design required for this approach. They also provide a precise language for specifications, enable clear comparisons between components, and support abstract evaluation of the capabilities of groups of differing components. These capabilities should powerfully encourage and support standardization.

Most difficult for compilers will be fully general shapes like the sculpted surfaces of automobiles. The vector and surface equations involved may (or may not) require substantial changes to the inference system. It may often be possible to simplify the inference process by parameterizing complex shapes with a few variables important to the rest of the design, then performing inferences on these parameters. Alternatively, RANGE, DOMAIN and SUFFICIENT-POINTS may be generalized to operate on volumes of N-space, rather than intervals on a line.

Multiple Component Operators

Each schematic symbol in the compiler's input language represents a clearly identified set of artifacts. Any operators which transform the meaning of the symbols act on only one symbol at a time; the system is hierarchically decomposed by virtue of the way it is constructed. The decomposition makes possible clear semantics; that is, we know what the design stands for. The clarity of the semantics, in turn, makes possible provably correct inferences.

Other work [26, 25] has involved operators separated from the artifact descriptions, which "recognize" the opportunity to transform a description. Such operators can work on more than one component at once; for example, they can recognize that the two ends of a not-yet-defined chain of artifacts should be joined by an artifact string of a certain kind. They therefore muddy the semantics of the design description.

We need to examine to what extent multi-component operators are pragmatically or theoretically needed. It they are, we need to examine whether they can be brought into hierarchical decomposition systems in ways that preserve clear abstraction semantics.

7.2.3 The Long Term: Outside Machine Design

Most of the following ideas lie outside my expertise. Some may in fact be fairly easy to address; others seem certain to be hard. They are intended to provide a feel for the possible applications of the "design calculus" outside the realms in which it was developed.

Large scale engineering

The general cycle of composition, abstraction, propagation, and elimination, and model refinement seems applicable on every scale; it may provide a rigorous description of the activities of engineering organizations. For example, based on past performance, it should be possible to formulate a description of the set of engines that an automobile engine design group might develop; this information could then be taken into account by the styling group during the idea generation phase.

To exploit this possibility the design calculus needs to be expanded and made yet more abstract, so that it can incorporate human activity, optimization, simulation, and probabilistic representations. This thesis has not addressed the most abstract aspects of the calculus, specifically its intersection with the model theory of logical semantics, but it appears that the set-based interpretation of the meaning of design descriptions may make possible a "Fundamental Theory of Design[1]." (Unfortunately, such a theory is unlikely to tell us much about the detailed process of design, just as the theory of evolution says little directly about molecular biology. But, such fundamental theories should tell us where to look for answers.)

The basic approach may be to treat abstraction, elimination, evaluation, performance proof and so on as "generic operations", able to function on a wide range of representations, including those in human heads. This is a substantial task, but it can probably be approached piecemeal, given a

[1] Yoshikawa [31] discusses an approach to fundamental issues based the topology of an "entity space" quite similar to our "artifact space".

solid theoretical foundation to ensure that the pieces fit together in the end. The ultimate goal should be a "development support" system in which every decision made throughout the organization is swiftly and automatically communicated to those it affects; authority is broadly distributed; and interdepartmental coordination (if not cooperation) is guaranteed because all the departments share the same model.

7.2.4 Planning

Considered abstractly, the quantitative inference system discusses sets of "things", which are described using variables, linked by equations. The variables can be limited to certain sets values, or can take on all the values in a set, or at least one of the values in the set. There seems no reason the "things" must be artifacts.

In particular, since the artifact is described only by its behavior in various sets of states, it may be possible in some domains to replace the connected artifacts with connected sets of states, yielding a planning system. This possibility is now being explored in the domain of assembly operation planning, where the objective is to take advantage of passive compliance in the assembly device (often a robot) and contacts between the part surfaces to guide the parts together.

Other planning applications might include the management of flexible manufacturing system, where the "things" would presumably be manufacturing tools. Rather than simulating system performance on a particular production mix, the planner might be able to consider sets of mixes, looking

for bottle-necks, or looking for the boundaries on efficiently producible mixes.

Similarly, military planners might be able to use quantitative models of units which take into account the tremendous uncertainties in enemy strength, leadership effectiveness, morale, and the effectiveness of fire. Financial planners might explicitly consider uncertainties in markets and interest rates, establishing boundaries on the actions certain to lead to successful performance.

Modeling, Diagnosis, and Sensor Fusion

Design and planning try to determine what *should be*; modeling, diagnosis, and sensor fusion operations try to determine what *is*. But there are important similarities. Uncertainty is the bane of quantitative models; it is usually incorporated using probability. But just as there is no probabilistic equivalent to such statements as "the torque will take on every value in the interval 0 to 200 newton-meters", there is no probabilistic equivalent to "a normal patient's heart rate will vary daily at least from 55 to 80." The quantitative inference mechanism provides a precise formulation for such statements, and a means of interpreting and manipulating them.

This might make possible more effective model-based diagnosis systems. "Components" of a quantitative human model would be physiological subsystems; "catalogs" would list versions of those subsystems as disturbed by various diseases. The model would propagate patient data, eliminating diseases that don't fit. The "cost function" for search would be based on the number of diseases required to account for the symptoms, as well as the

likelihood of those diseases; the "optimal solution" would be the "most reasonable" interpretation of the results.

"Sensor fusion" problems, such as determining the identity of an enemy anti-aircraft system, seem essentially similar.

Predictive models, say of the global economy, may also benefit from explicit recognition that we face a set of possible futures, and that there is a set of possible human behaviors. Further, cause and effect are rarely precisely assignable; even after events have occurred we can rarely unambiguously decide which model was appropriate.

An approach might be to formulate a set of competing models for each sub-system of the economy, then run the composite model on collected data, eliminating inconsistent component models. In this case, we probably don't want an "optimal model"; we want to know the range of plausible futures.

Finally, if we can do "large scale engineering", perhaps we can do "large scale science". In dealing with such complex objects as biological systems (or the economy), it is hard to check the models of parts of the system for consistency with the system as a whole. If we can catalog the alternative models for subsystems, and identify the constraints these models impose on each other, we can perhaps establish automatic communication mechanisms between specialists. These might significantly aid us in formulating models of systems too complex for any single person to understand.

Bibliography

[1] David Serrano and David Gossard. Constraint management in conceptual design. In *Knowledge Based Expert Systems in Engineering, Planning and Design.* Computational Mechanics Publications, 1987.

[2] R. J. Popplestone. The Edinburgh Designer system as a framework for robotics: the design of behavior. *Artifical Intelligence for Engineering Design, Analysis, and Manufacturing*, 1(1), 1987.

[3] Kenneth L. Meunier and John R. Dixon. Iterative respecification: A computational model of hierarchical mechanical system design. In *ASME Computers in Engineering Conference.* ASME, 1988.

[4] Jack Mostow, Lou Steinberg, Noshir Langrana, and Chris Tong. A domain-independent model of knowledge-based design: Progress report to the National Science Foundation. Technical Report Working Paper 90-1, Rutgers University, 1988.

[5] Allen C. Ward and Warren Seering. An approach to computational aids for mechanical design. In *Proc. 1987 International Conference on Engineering Design.* ASME, 1987.

[6] Chaim D. Shin-Orr. *Automatic Design of Complex Gear Trains.* PhD thesis, Massachusetts Institute of Technology, 1976.

[7] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32, 1987.

[8] David Serrano. *Constraint Management in Conceptual Design.* PhD thesis, Massachusetts Institute of Technology, 1989.

[9] Alan Fleming. Analysis of uncertainties in a structure of parts. In *International Joint Conference on Artificial Intelligence*, 1985.

[10] J. R. Dixon, E. C. Libardi Jr., S. C. Luby, M. Vaghul, and M. K. Simmons. Expert systems for mechanical design: Examples of symbolic representations of design geometries. In *Applications of Knowledge-based Systems to Engineering Analysis and Design.* ASME, New York, NY, 1985.

[11] Gerald Jay Sussman and Guy Lewis Steele. Constraints—a language for expressing almost heirarchical descriptions. *Artificial Intelligence*, 14, 1980.

[12] Mark Donald Gross. *Design as Exploring Constraints.* PhD thesis, Massachusetts Institute of Technology, 1985.

[13] Panos Y. Papalambros. The design laboratory: Interdisciplinary research and education. In *Proceedings of the 1988 NSF Grantee Workshop on Design Theory and Methodology*, Rensselaer Polytechnic Institute, Troy, NY, 1988.

[14] S. Mittal, C. L. Dym, and M. Morjaria. PRIDE: An expert system for the design of paper paths. In *Applications of Knowledge-based Systems to Engineering Analysis and Design.* ASME, New York, NY, 1985.

[15] Walter Hamscher. Using structural and functional information in diagnostic design. Technical Report 707, MIT Artificial Intelligence Laboratory, June 1983.

[16] David Chapman. Planning for conjuntive goals. Technical Report 802, MIT Artificial Intelligence Lab, 1985.

[17] Aristides A. G. Requicha. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4), 1983.

[18] Daniel S. Weld. The use of aggregation in qualitative simulation. *Artificial Intelligence*, 30, 1988.

[19] Kristin L. Wood and Erik K. Antonsson. Comparing fussy and probability calculus for representing imprecision in preliminary engineering design. In *First International ASME Design Theory and Methodology Conference*, Submitted 1989.

[20] David Brown. Capturing mechanical design knowledge. In *Proceedings of the 1985 ASME International Computers in Engineering Conference*, Boston, MA, 1985. ASME.

[21] M. F. Orelup, J. R. Dixon, and M. K. Simmons. Dominic II: More progress toward domain independent design by iterative redesign. In *Proceedings of the ASME Winter Annual Meeting.* ASME, 1987.

[22] Gerald Jay Sussman and Harold Abelson. *Structure and Interpretation of Computer Programs.* The MIT Press, 1985.

[23] Allen C. Ward and Warren Seering. Representing component types for design. In *Advances in Design Automation—1987.* ASME, 1987.

[24] Karl T. Ulrich and Warren P. Seering. Conceptual design: Synthesis of novel systems of components. In *Proceedings of the 1987 ASME Winter Annual Meeting—Symposium on Intelligent and Integrated Manufacturing Analysis and Synthesis.* ASME, 1987.

[25] Leo Joskowicz. Reasoning about shape and kinematic function in mechanical devices. Technical Report 402, Courant Institute of Mathematical Sciences, 1988.

[26] Karl T. Ulrich. *Computation and Pre-Parametric Design.* PhD thesis, Massachusetts Institute of Technology, 1988.

[27] Johan de Kleer. Qualitative and quantitative knowledge in classical mechanics. Master's thesis, Massachusetts Institute of Technology, 1975.

[28] I. Sutherland. Sketchpad—a man-machine graphical communication system. Technical Report 296, Lincoln Laboratory, Massachusetts Institute of Technology, 1963.

[29] A. Jerraya, P. Varinot, R. Jamier, and B. Courtois. Principles of the SYCO compiler. In *Proceedings of the 23rd Design Automation Conference.* IEEE, 1986.

[30] Patrick Fitzhorn. A computational theory of design. *Design Computing*, 3(1), 1988.

[31] H. Yoshikawa. *Design Theory.* University of Tokyo, 1986.

[32] M. B. Waldron. Modeling of the design process. In *Proceedings of the 1988 IFIP 5.2 Intelligent CAD Workshop.* North-Holland, 1988.

[33] David G. Ullman and Thomas G. Dietterich. Mechanical design methodology: Implications for future developments of computer-aided design and knowledge-based systems. *Engineering with Computers*, 2, 1987.

[34] James R. Rinderle, Eric R. Colburn, Stephen P. Hoover, Juan Pedro Paz-Soldan, and John D. Watton. Form-function characteristics of mechanical designs—research in progress. In *Proceedings of the 1988 NSF Grantee Workshop on Design Theory and Methodology*, Rensselaer Polytechnic Institute, Troy, NY, 1988.

[35] Kristin L. Wood and Erik K. Antonsson. Computations with imprecise parameters in engineering design: Background and theory. Technical Report 88-01, Engineering Design Research Laboratory, California Institute of Technology, 1988.

[36] Kristin L. Wood and Erik K. Antonsson. Computations with imprecise parameters in engineering design: Applications and examples. Technical Report 88-02, Engineering Design Research Laboratory, California Institute of Technology, 1988.

[37] Jonathan Cagan and Alice M. Agogino. Innovative design of mechanical structures from first principles. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 1988.

[38] Tomás Lozano-Pérez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[39] Genichi Taguchi. *Introduction to Quality Engineering.* UNIPUB, White Plains, NY, 1986.

Appendix A

Definitions

1. $\langle \overset{only}{[\ ]}\ x\ X \rangle$: The variable x takes on only values in X.
2. $\langle \overset{every}{\leftrightarrow}\ x\ X \rangle$: The variable x takes on every value in X.
3. $\langle \overset{some}{\cdots}\ x\ X \rangle$: The variable x takes on some value in X.
4. $\langle \overset{none}{\bowtie}\ x\ X \rangle$: The variable x takes on no values in X.
5. Assured (**A**): The statement is true for every artifact.
6. Required (**R**): The statement must be true for a satisfactory design.
7. No-stronger (**N**): No stronger statement is possible.
8. RANGE(G,X,Y): Z, the possible values of $g(x, y)$, with x in X and y in Y.
9. DOMAIN(G,Z,X): Y, such that RANGE$(G, X, Y) = Z$
10. SUFPT(G,Z,X): Y, the possible values of y such that RANGE$(G, X, y) \supseteq Z$

Table A.1: Informal Definitions

1. $\langle {}^{only}_{[\ ]}\ X\rangle \stackrel{def}{\Longleftrightarrow} \forall s \in S, \exists \mathbf{x} \in X.\mathbf{x}(s) = \mathbf{x}$

2. $\langle {}^{every}_{\leftrightarrow}\ X\rangle \stackrel{def}{\Longleftrightarrow} \forall \mathbf{x} \in X, \exists s \in S.\mathbf{x}(s) = \mathbf{x}$

3. $\langle {}^{some}_{\cdots}\ X\rangle \stackrel{def}{\Longleftrightarrow} \exists s \in S.\exists \mathbf{x} \in X.\mathbf{x}(s) = \mathbf{x}$

4. $\langle {}^{none}_{\bowtie}\ X\rangle \stackrel{def}{\Longleftrightarrow} \forall s \in S.\mathbf{x}(s) \notin X$

5. **Assured:** $\mathbf{A}(p, \mathrm{C}, S) \stackrel{def}{\Longleftrightarrow} \forall a \in \mathrm{C}, p(a, S)$

6. **Required:**
   $\mathbf{R}(p, \mathrm{C}, S) \stackrel{def}{\Longleftrightarrow} \forall A \in \mathrm{C}, \exists a \in A.\neg p(a, S)) \longrightarrow$ UNSATISFACTORY(a)

7. $\mathbf{N}(\langle {}^{only}_{[\ ]}\ X\rangle, \mathrm{C}, S) \stackrel{def}{\Longleftrightarrow} \forall A \in \mathrm{C}, \forall \mathbf{x} \in X, \exists a \in A.\exists s \in S.\mathbf{x}(a, s) = \mathbf{x}$

8. $\mathbf{N}(\langle {}^{every}_{\leftrightarrow}\ X\rangle, \mathrm{C}, S) \stackrel{def}{\Longleftrightarrow}$
   $\forall A \in \mathrm{C},$
   $[\exists a \in A.\forall s \in S.\mathbf{x}(a, s) \leq x_h \& \exists a \in A.\forall s \in S.\mathbf{x}(a, s) \geq x_l$

9. RANGE$(G, X, Y) = \{\mathbf{z} | \exists \mathbf{x} \in X, \exists \mathbf{y} \in Y.G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

10. CORNERS$(G, X, Y) = \{g(\mathbf{x}_l, \mathbf{y}_l), g(\mathbf{x}_h, \mathbf{y}_l), g(\mathbf{x}_l, \mathbf{y}_h), g(\mathbf{x}_h, \mathbf{y}_h)\}$

11. DOMAIN(G, Z, X) $= Y$ **iff** RANGE$(G, X, Y) = Z$
    $= \{y | \forall \mathbf{x} \in X, \exists \mathbf{z} \in Z.G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

12. SUFPT(G, Z, X) $= \{\mathbf{y} | Z \subseteq$ RANGE$(G, X, \mathbf{y})\}$
    $= \{\mathbf{y} | \forall \mathbf{z} \in Z, \exists \mathbf{x} \in X.G(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0\}$

13. STATE-CONTINOUS: Let $\mathbf{x}(s_1) < \mathbf{x} < \mathbf{x}(s_2)$ for some $s_1, s_2 \in S$; if this implies that there exists some $s \in S$ such that $\mathbf{x} = \mathbf{x}(s)$, then $\mathbf{x}$ is STATE-CONTINUOUS.

14. PARAMETER(x) if and only if there is some single assignment $\mathbf{x}_0$ such that for all $s \in S$, $\mathbf{x}(s) = \mathbf{x}_0$.

Table A.2: Formal Definitions

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 8/3/95

Report # AI-TR-1089

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 146 (154-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form (2) ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): PAGE FOLLOWING TITLE, 2, 7

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP (1-10) UN#'ED TITLE, BLANK, 2, UN# BLANK, 3-7, UN#'D BLANK
(11-146) PAGES #'ED 8-100, UN#'ED BLANK, 101-137,
UN#'ED PAGE, 139, 2 UN#'ED PAGES, UN#'ED BLANK
(147-154) SCAN CONTROL, COVER, SPINE, DOD(2), TRGT (3)

Scanning Agent Signoff:

Date Received: 8/3/95 Date Scanned: 8/14/95 Date Returned: 8/17/95

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR 1089	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Theory of Quantitative Inference Applied to a Mechanical Design Compiler		5. TYPE OF REPORT & PERIOD COVERED technical report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Allen C. Ward		8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0685
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

mechanical design
constraint propagation
quantitative reasoning
labeled internal calculus

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis presents the ideas underlying a computer program that takes as input a schematic of a mechanical or hydraulic power transmission system, plus specifications and a utility function, and returns catalog numbers from predefined catalogs for the optimal selection of components implementing the design. Unlike programs for designing single components or systems, the program provides the designer with a high level "language" in which to compose new designs. It then performs some of the detailed design process. (cont. on back)

Block 20 cont.

The process of "compilation", or transformation from a high to a low level description, is based on a formalization of quantitative inferences about hierarchically organized sets of artifacts and operating conditions. This allows design compilation without the exhaustive enumeration of alternatives.

The program has been tested on a wide variety of power transmission and temperature sensing problems. Key elements of the theory have been formally proven. It appears that the theory has applications outside design.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 8/14/1995

M.I.T. Libraries
Document Services